Schematic Generation User Guide

Daniel Reyno

May 8, 2011

1. Setup

To get started with schematic generation, you first need to download the schematic generation script package. The script package can be found on the Wiki under the ViPro page and the .zip file is called "darViPro.zip". This script package contains 11 SKILL files (*.il) which are needed for complete functionality of the automatic schematic generation. All SKILL files must be placed within a subdirectory called 'skill' located in the folder where you run Cadence, i.e. all files must be located in "/cadence/skill/". Furthermore, the schematic generation requires three text files that contain input parameters for the memory. Unlike the SKILL files, these three files can be located anywhere within the user's account; they do not need to be placed within the /skill subfolder. The three text files have specific purposes. One text file contains the minimum FET sizes for the given technology and the file paths to the other two text files. Another text file contains the architecture specifications for the desired memory. The final text file contains parameters for the memory, such as bitcell sizing, the number of stages in certain buffer chains, the fanout of certain buffer chains, etc.

2. Operation

Before you can automatically generate the schematics of the memory, you must first specify some input parameters located within the three mentioned text files. Every text file must have a specific format in order for the procedures to work properly. Each line within the text file must contain 2 data values, a parameter name and a parameter value. Because these procedures use hash maps (actually called 'association tables' in SKILL) to store the data, the parameter name serves as the key and the parameter value is the value. The parameter name must follow a specific naming convention in order for the procedures to function correctly. Finally, all text files must contain a single blank line at the end of the file in order for the procedures to work correctly.

The first text file must contain the minimum FET sizes for the given technology node and the file paths to the other two text files. Thus, there are 6 keys within this text file; there is no required order for these keys, only that all keys exist within the file. The required naming convention is as follows: wpMin, lpMin, wnMin, lnMin, inputsPath, and sizesPath. The FET sizes names are pretty straightforward. The inputsPath is the file path to the text file that contains architecture specificiations, i.e. number of rows, number of columns, word length etc. The sizesPath is the file path to the text file that contains sizing parameters for the memory, i.e. bitcell sizing, buffer chain fanouts, etc. Each of these keys, which are on separate lines, is followed by their desired value on the same line. For the FET sizes, they must be in abbreviated units, i.e. for nano, use n, for micro, use u, etc. For the file paths, you can specify either the full file path or the file path relative to the folder where you run Cadence. Providing the full file path will ensure no errors.

A sample of this file is as follows (the actual text file is the information in between the --- marks):

```
---------
wpMin 180n
lpMin 50n
wnMin 90n
lnMin 50n
inputsPath ./skill/inputs.txt
sizesPath ./skill/sizes.txt
(required blank line)
---------
```

The second text file must contain the architecture specifications for the desired memory. There are 9 keys within this text file; there is no required order for these keys, only that all keys exist within the file. The required naming convention is as follows: libName, technologyCase, FETlibName, PMOScellName, NMOScellName, rows, addrBits, columns, wordSize. Each of these keys, which are on separate lines, is followed by their desired value on the same line. The keys provide the following purposes:

libName – String specifying the library name to place the schematic
technologyCase – String specifying the required letter case for the given technology node; for example, for the 45nm FreePDK, letter case is upper, for ST65nm, letter case is lower
FETlibName – String specifying the library where the FETs to be used are located
PMOScellName – String specifying the PMOS cell name to use
NMOScellName – String specifying the NMOS cell name to use
rows – Integer specifying the number of rows in the memory
addrBits – Integer specifying the number of address bits; currently, this must always be 9
columns – Integer specifying the number of columns in the memory
wordSize – Integer specifying the length of a word in the memory

A sample of this file is as follows (the actual text file is the information in between the --- marks):

```
---------
libName ViPro
technologyCase upper
FETlibName NCSU_Devices_FreePDK45
PMOScellName PMOS_VTL
NMOScellName NMOS_VTL
rows 64
addrBits 9
columns 8
wordSize 1
(required blank line)
---------
```

The final text file must contain the sizing parameters for the memory. There are 60 possible keys within this text file and 20 of them are required; there is no required order for these keys, only that the 20 essential ones exist. Each of these keys, which are on separate lines, is followed by their desired value on the same line. The required naming convention is divided into each circuits/gate and is as follows:

Bitcell
wpg, lpg, wpd, lpd, wpu, lpu

N/K Buffer Chains
Nstages, NFanOut
Kstages, KFanOut

Timing Block
dlyWLstages, dlyWLFanOut
ICLKstages, ICLKFanOut
NSAPRECstages, NSAPRECFanOut
WLENstages, WLENFanOut
SAEstages, SAEFanOut
WENstages, WENFanOut
NPRECHstages, NPRECHFanOut

Sense Amp
SAwen, SAlen, SAweql, SAleql, SAwpsa, SAlpsa, SAwnsa, SAlnsa, SAwbl, SAlbl, SAwsapc, SAlsapc, SAwpNAND, SAlpNAND, SAwnNAND, SAlnNAND

Column DEMUX
CDwpc, CDlpc, CDwpTX, CDlpTX, CDwnTX, CDlnTX

IOBlock
TSIwpINV, TSIwnINV, TSIwpEN, TSIwnEN, TSIlpINV, TSIlnINV, TSIlpEN, TSIlnEN
IOwpINV, IOwnINV, IOlpINV, IOlnINV

Top Level Timing
CSELstages, CSELFanOut

The required keys are:
Nstages, NFanOut
Kstages, KFanOut
dlyWLstages, dlyWLFanOut
ICLKstages, ICLKFanOut
NSAPRECstages, NSAPRECFanOut
WLENstages, WLENFanOut
SAEstages, SAEFanOut
WENstages, WENFanOut
NPRECHstages, NPRECHFanOut
CSELstages, CSELFanOut

The required keys are the number of stages and the fanout for all of the buffer chains within the memory. If even a single one of these keys is not set to some value, then the schematics won't automatically generate and SKILL will output an error. Any of the optional keys (all of the optional keys are for FET sizes) which are not specified will set FET sizes to the minimum value

for the technology node. A sample of this file is as follows (the actual text file is the information in between the --- marks):

```
---------
wpg 720n
wpd 500n
lpg 73n
CDwpc 83n
CDwnTX 125n
dlyWLstages 4
dlyWLFanOut 10
ICLKstages 4
ICLKFanOut 8
NSAPRECstages 6
NSAPRECFanOut 4
WLENstages 8
WLENFanOut 2
SAEstages 10
SAEFanOut 6
WENstages 12
WENFanOut 4
NPRECHstages 14
NPRECHFanOut 2
CSELstages 8
CSELFanOut 2
Nstages 0
NFanOut 1.5
Kstages 6
KFanOut 4
TSIwnINV 200n
TSIwpEN 355n
SAwen 207n
SAwnNAND 185n
SAweql 195n
IOwpINV 187n
IOwnINV 93n
(required blank line)
---------
```

All required keys have been bolded and italicized.

Once all SKILL files are placed in the /skill directory and all three text files have been placed, there is one more step before you can automatically generate the entire schematic. Within the file “darUvaEceSRAM_Schem.il”, there are lines of code near the top of the file that say:

```
;;;;-----------------------------------
;;;; MUST store minimum sizes in association
;;;; table before loading any SKILL file

  ;; read the default FET sizes for the given technology
  minSizeTable = storeData("./skill/minSizes.txt")
```

Within the argument of the storeData() command, you must specify the file path (either full or relative to the Cadence directory) to the text file that contains the minimum FET sizes and the other two files paths. In the example presented above, this text file is called ‘minSizes.txt’ and it is located within the subfolder called ‘skill’ within the directory where Cadence is run from.

Once you input the file path to the function, you are ready to automatically generate the complete schematics.

There are two ways to automatically generate the complete schematics. One way is to start Cadence and within the CIW window, input the following command: load "./skill/darUvaEceSRAM_Schem.il". The quotations are required around the file path. Another way is to load the top-level SKILL file through ViPro. See the ViPro User Guide for more information on this process.

3. Circuit Schematics

The circuit schematics for the individual components can be found at the end of this user guide, after the API documentation.

4. API Documentation

A complete API-style documentation of all the procedures used to create the entire SRAM schematic can be found directly following this section.

darSchematic.il

* All of the following procedures check the generated schematic, then create the symbol from the schematic, unless otherwise noted.
* VDD and VSS are always an input/output pin in every circuit.
* All parameters are read in as strings, unless otherwise noted.
* All examples assume the use of the 45nm Free PDK, which requires uppercase letters for the FET terminals. Thus, the parameter "technologyCase" will always be "upper" for each example.

darCreateInverterSchematic()

```
darCreateInverterSchematic(
        libName
        cellName
        FETlibName
        PMOScellName
        NMOScellName
        technologyCase
        )
```

Description

Creates a parameterized inverter with an input pin (IN) and an output pin (OUT). The created inverter has the following parameters: wp, lp, wn, ln.

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
FETlibName	String specifying the library where the FETs are located.
PMOScellName	String specifying the cell name of the PMOS FET to be placed.
NMOScellName	String specifying the cell name of the NMOS FET to be placed.
technologyCase	String specifying the letter required for the used technology.

Example

darCreateInverterSchematic("ViPro" "INV" "NCSU_Devices_FreePDK45" "PMOS_VTL" "NMOS_VTH" "upper")

=> Creates an inverter called INV in the ViPro libary by placing a PMOS_VTL device and NMOS_VTH device from the NCSU_Devices_FreePDK45 library.

darCreateParInverter()

darCreateParInverter(
libName
cellName
wp
wn
lp
ln
)

Description
Parameterizes an inverter by creating CDF Parameters for the following parameters: wp, wn, lp, ln. This procedure is normally called within darCreateInverterSchematic().

Arguments

libName	String specifying the library where the inverter to be parameterized is located.
cellName	String specifying the cell name of the inverter to be parameterized.
wp	String that gives a default PMOS width. Usually the minimum device size for the given technology.
wn	String that gives a default NMOS width. Usually the minimum device size for the given technology.
lp	String that gives a default PMOS length. Usually the minimum device size for the given technology.
ln	String that gives a default NMOS width. Usually the minimum device size for the given technology.

Examples
darCreateParInverter(“ViPro” “INV” “180n” “90n” “50n” “50n”)
=> Returns INV as a parameterized inverter with the following default values: PMOS width = 180n, NMOS width = 90n, PMOS legnth = 50n, and NMOS length = 50n.

darCreateBufferSchematic()

darCreateBufferSchematic(
libName
cellName
FETlibName
PMOScellName
NMOScellName
technologyCase
)

Description
Creates a parameterized buffer with an input pin (IN) and an output pin (OUT). The created buffer has the following parameters: wp1, lp1, wn1, ln1, wp1, lp1, wn1, and ln1. The 1 denotes the parameters for the first inverter while the 2 denotes the parameters for the second inverter.

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
FETlibName	String specifying the library where the FETs are located.
PMOScellName	String specifying the cell name of the PMOS FET to be placed.
NMOScellName	String specifying the cell name of the NMOS FET to be placed.
technologyCase	String specifying the letter required for the used technology.

Example
darCreateBufferSchematic(“ViPro” “BUFF” “NCSU_Devices_FreePDK45” “PMOS_VTL” “NMOS_VTL “upper”)
=> Creates an buffer called BUFF in the ViPro libary by placing PMOS_VTL devices and NMOS_VTL devices from the NCSU_Devices_FreePDK45 library.

darCreateParBuffer()

```
darCreateParBuffer(
        libName
        cellName
        wp1
        wn1
        lp1
        ln1
        wp2
        wn2
        lp2
        ln2
        )
```

Description
Parameterizes a buffer by creating CDF Parameters for the following parameters: wp1, lp1, wn1, ln1, wp1, lp1, wn1, and ln1. This procedure is normally called within darCreateBufferSchematic().

Arguments

libName	String specifying the library where the buffer to be parameterized is located.
cellName	String specifying the cell name of the buffer to be parameterized.
wp1	String that gives a default PMOS width. Usually the minimum device size for the given technology.
wn1	String that gives a default NMOS width. Usually the minimum device size for the given technology.
lp1	String that gives a default PMOS length. Usually the minimum device size for the given technology.
ln1	String that gives a default NMOS width. Usually the minimum device size for the given technology.
wp2	String that gives a default PMOS width. Usually the minimum device size for the given technology.
wn2	String that gives a default NMOS width. Usually the minimum device size for the given technology.
lp2	String that gives a default PMOS length. Usually the minimum device size for the given technology.

ln2	String that gives a default NMOS width. Usually the minimum device size for the given technology.

Example

darCreateParBuffer(“ViPro” “BUFF” “180n” “90n” “50n” “50n” “275n” “125n” “50n” “50n”) => Returns BUFF as a parameterized buffer with the following default values: PMOS1 width = 180n, NMOS1 width = 90n, PMOS1 legnth = 50n, NMOS1 length = 50n, PMOS2 width = 275n, NMOS2 width = 125n, PMOS2 legnth = 50n, and NMOS2 length = 50n.

darCreateNAND2Schematic()

darCreateNAND2Schematic(
libName
cellName
FETlibName
PMOScellName
NMOScellName
technologyCase
)

Description
Creates a parameterized 2-input NAND gate with an input pin (IN) and an output pin (OUT). The created NAND gate has the following parameters: wp, lp, wn, and ln.

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
FETlibName	String specifying the library where the FETs are located.
PMOScellName	String specifying the cell name of the PMOS FET to be placed.
NMOScellName	String specifying the cell name of the NMOS FET to be placed.
technologyCase	String specifying the letter required for the used technology.

Examples
darCreateNAND2Schematic(“ViPro” “NAND2” “NCSU_Devices_FreePDK45” “PMOS_VTH” “NMOS_VTL “upper”)
=> Creates a 2-input NAND gate called NAND2 in the ViPro libary by placing PMOS_VTH devices and NMOS_VTL devices from the NCSU_Devices_FreePDK45 library.

darCreateParNand2()

darCreateParNand2(
libName
cellName
wp
wn
lp
ln
)

Description
Creates a parameterized 2-input NAND gate with an input pin (IN) and an output pin (OUT). The created NAND gate has the following parameters: wp, lp, wn, and ln. This procedure is normally called within darCreateNAND2Schematic().

Arguments

libName	String specifying the library where the inverter to be parameterized is located.
cellName	String specifying the cell name of the NAND gate to be parameterized.
wp	String that gives a default PMOS width. Usually the minimum device size for the given technology.
wn	String that gives a default NMOS width. Usually the minimum device size for the given technology.
lp	String that gives a default PMOS length. Usually the minimum device size for the given technology.
ln	String that gives a default NMOS width. Usually the minimum device size for the given technology.

Example
darCreateParNand2(“ViPro” “NAND2” “180n” “90n” “50n” “50n”)
=> Returns NAND2 as a parameterized 2-input NAND gate with the following default values: PMOS width = 180n, NMOS width = 90n, PMOS legnth = 50n, and NMOS length = 50n.

darCreateAND2Schematic()

darCreateAND2Schematic(
libName
cellName
FETlibName
PMOScellName
NMOScellName
technologyCase
)

Description
Creates a parameterized 2-input aAND gate with an input pin (IN) and an output pin (OUT). The created AND gate has the following parameters: wpNAND, lpNAND, wnNAND, lnNAND, wpINV, lpINV, wnINV, and lnINV.

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
FETlibName	String specifying the library where the FETs are located.
PMOScellName	String specifying the cell name of the PMOS FET to be placed.
NMOScellName	String specifying the cell name of the NMOS FET to be placed.
technologyCase	String specifying the letter required for the used technology.

Examples
darCreateAND2Schematic(“ViPro” “AND2” “NCSU_Devices_FreePDK45” “PMOS_VTH” “NMOS_VTL “upper”)
=> Creates a 2-input NAND gate called NAND2 in the ViPro libary by placing PMOS_VTH devices and NMOS_VTL devices from the NCSU_Devices_FreePDK45 library.

darCreateParAnd2()

```
darCreateParAnd2(
        libName
        cellName
        wpNAND
        wnNAND
        lpNAND
        lnNAND
        wpINV
        wnINV
        lpINV
        lnINV
        )
```

Description

Creates a parameterized 2-input AND gate with an input pin (IN) and an output pin (OUT). The created AND gate has the following parameters: wpNAND, lpNAND, wnNAND, lnNAND, wpINV, lpINV, wnINV, and lnINV. This procedure is normally called within darCreateAND2Schematic().

Arguments

libName	String specifying the library where the inverter to be parameterized is located.
cellName	String specifying the cell name of the AND gate to be parameterized.
wpNAND	String that gives a default PMOS width of the NAND portion of the AND gate. Usually the minimum device size for the given technology.
wnNAND	String that gives a default NMOS width of the NAND portion of the AND gate. Usually the minimum device size for the given technology.
lpNAND	String that gives a default PMOS length of the NAND portion of the AND gate. Usually the minimum device size for the given technology.
lnNAND	String that gives a default NMOS width of the NAND portion of the AND gate. Usually the minimum device size for the given technology.
wpINV	String that gives a default PMOS width of the inverter portion of the AND gate. Usually the minimum device size for the given technology.
wnNAND	String that gives a default NMOS width of the inverter portion of the AND gate. Usually the minimum device size for the given technology.
lpNAND	String that gives a default PMOS length of the inverter portion of the

	AND gate. Usually the minimum device size for the given technology.
lnNAND	String that gives a default NMOS width of the inverter portion of the AND gate. Usually the minimum device size for the given technology.

Example

darCreateParAnd2(“ViPro” “AND2” “180n” “90n” “50n” “50n” “200n” “125n” “50n” “50n”) => Returns AND2 as a parameterized 2-input AND gate with the following default values: PMOS width of NAND portion = 180n, NMOS width of NAND portion = 90n, PMOS legnth of NAND portion = 50n, NMOS length of NAND portion = 50n, PMOS width of inverter portion = 200n, NMOS width of inverter portion = 125n, PMOS legnth of inverter portion = 50n, NMOS length of inverter portion = 50n.

darCreateNOR2Schematic()

darCreateNOR2Schematic(
libName
cellName
FETlibName
PMOScellName
NMOScellName
technologyCase
)

Description
Creates a parameterized 2-input NOR gate with an input pin (IN) and an output pin (OUT). The created NAND gate has the following parameters: wp, lp, wn, and ln.

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
FETlibName	String specifying the library where the FETs are located.
PMOScellName	String specifying the cell name of the PMOS FET to be placed.
NMOScellName	String specifying the cell name of the NMOS FET to be placed.
technologyCase	String specifying the letter required for the used technology.

Examples
darCreateNOR2Schematic(“ViPro” “NOR2” “NCSU_Devices_FreePDK45” “PMOS_VTH” “NMOS_VTL “upper”)
=> Creates a 2-input NOR gate called NOR2 in the ViPro libary by placing PMOS_VTH devices and NMOS_VTL devices from the NCSU_Devices_FreePDK45 library.

darCreateParNor2()

darCreateParNor2(
libName
cellName
wp
wn
lp
ln
)

Description
Creates a parameterized 2-input NOR gate with an input pin (IN) and an output pin (OUT). The created NOR gate has the following parameters: wp, lp, wn, and ln. This procedure is normally called within darCreateNOR2Schematic().

Arguments

libName	String specifying the library where the inverter to be parameterized is located.
cellName	String specifying the cell name of the NOR gate to be parameterized.
wp	String that gives a default PMOS width. Usually the minimum device size for the given technology.
wn	String that gives a default NMOS width. Usually the minimum device size for the given technology.
lp	String that gives a default PMOS length. Usually the minimum device size for the given technology.
ln	String that gives a default NMOS width. Usually the minimum device size for the given technology.

Example
darCreateParNor2(“ViPro” “NOR2” “180n” “90n” “50n” “50n”)
=> Returns NOR2 as a parameterized 2-input NOR gate with the following default values:
PMOS width = 180n, NMOS width = 90n, PMOS legnth = 50n, and NMOS length = 50n.

darCreateTriStateInverterSchematic()

darCreateTriStateInverterSchematic(
libName
cellName
FETlibName
PMOScellName
NMOScellName
technologyCase
)

Description
Creates a parameterized tri-state inverter with an input pin (IN), enable pins (TRIP for PMOS enable FET and TRIN for NMOS enable FET) and an output pin (OUT). The created tri-state inverter has the following parameters: wpINV, wpEN, wnEN, wnINV, lpINV, lpEN, lnEN, and lnINV.

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell
FETlibName	String specifying the library where the FETs are located.
PMOScellName	String specifying the cell name of the PMOS FET to be placed.
NMOScellName	String specifying the cell name of the NMOS FET to be placed.
technologyCase	String specifying the letter required for the used technology.

Example
darCreateTriStateInverterSchematic(“ViPro” “TSI” “NCSU_Devices_FreePDK45” “PMOS_VTH” “NMOS_VTL “upper”)
=> Creates a tri-state inverter called TSI in the ViPro libary by placing PMOS_VTH devices and NMOS_VTL devices from the NCSU_Devices_FreePDK45 library.

darCreateParTriStateInverter()

```
darCreateParTriStateInverter(
        libName
        cellName
        wpINV
        wpEN
        wnEN
        wnINV
        lpINV
        lpEN
        lnEN
        lnINV
        )
```

Description
Parameterizes a tri-state inverter by creating CDF Parameters for the following parameters: wpINV, wpEN, wnEN, wnINV, lpINV, lpEN, lnEN, and lnINV. This procedure is normally called within darCreateTriStateInverterSchematic().

Arguments

libName	String specifying the library where the tri-state inveter to be parameterized is located.
cellName	String specifying the cell name of the inverter to be parameterized.
wpINV	String that gives a default PMOS width of the inverter portion of the tri-state inverter. Usually the minimum device size for the given technology.
wnINV	String that gives a default NMOS width of the inverter portion of the tri-state inverter. Usually the minimum device size for the given technology.
lpINV	String that gives a default PMOS length of the inveter portion of the tri-state inverter. Usually the minimum device size for the given technology.
lnINV	String that gives a default NMOS width of the inverter portion of the tri-state inverter.. Usually the minimum device size for the given technology.
wpEN	String that gives a default PMOS width of the enable portion of the tri-state inverter. Usually the minimum device size for the given technology.

wnEN	String that gives a default NMOS width of the enable portion of the tri-state inverter. Usually the minimum device size for the given technology.
lpEN	String that gives a default PMOS length of the enable portion of the tri-state inverter. Usually the minimum device size for the given technology.
lnEN	String that gives a default NMOS width of the enable portion of the tri-state inverter. Usually the minimum device size for the given technology.

Example
darCreateParTriStateInverter(“ViPro” “TSI” “360n” “360n” “180n” 180n” “50n” “50n” “50n” “50n”)
=> Returns TSI as a parameterized tri-state inverter with the following default values: Inverter PMOS width = 360n, Enable PMOS width = 360n, Enable NMOS width = 180n, Inverter NMOS width = 180n, Inverter PMOS length = 50n, Enable PMOS length = 50n, Enable NMOS length = 50n, Inverter NMOS length = 50n

darCreateBitcellSchematic()

```
darCreateBitcellSchematic(
        libName
        cellName
        FETlibName
        PMOScellName
        NMOScellName
        technologyCase
        @key (wpg wnMin) (lpg lnMin) (wpu wpMin) (lpu lpMin) (wpd wnMin) (lpd lnMin)
        )
```

Description

Creates a typical 6 transistor bitcell with an input word line pin (WL) and input/ouput bit line pins (BL/BLB). FET sizes are set to default which is the minimum device sizes for the given technology. Users can alter any of the 6 FET widths and/or lengths by changing the desired parameter. The bitcell has the following parameters: wpg, lpg, wpu, lpu, wpd, and lpd.

The abbreviations correspond to the following FETs in the schematic:
pu - pull up transistors (PMOS)
pd - pull down transistors (PMOS)
pg - pass gate/access transistors (NMOS)

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
FETlibName	String specifying the library where the FETs are located.
PMOScellName	String specifying the cell name of the PMOS FET to be placed.
NMOScellName	String specifying the cell name of the NMOS FET to be placed.
technologyCase	String specifying the letter required for the used technology.
wpg	String specifying the width of the pass gate (NMOS) FETs.
lpg	String specifying the legnth of the pass gate (NMOS) FETs.
wpu	String specifying the width of the pull up (PMOS) FETs.
lpu	String specifying the width of the pull up (PMOS) FETs.
wpd	String specifying the width of the pull down (NMOS) FETs.

lpd String specifying the width of the pull down (NMOS) FETs.

Example
darCreateBitcellSchematic(“ViPro” “BCmin” “NCSU_Devices_FreePDK45” “PMOS_VTL” “NMOS_VTL “upper”)
=> Creates a 6T bitcell called BCmin in the ViPro libary by placing PMOS_VTL devices and NMOS_VTL devices from the NCSU_Devices_FreePDK45 library. The bitcell will have minimum device sizing because no parameters were specified.

darCreateBitcellSchematic(“ViPro” “BC” “NCSU_Devices_FreePDK45” “PMOS_VTL” “NMOS_VTL “upper” ?wpg “570n” ?wpu “220n”)
=> Creates a 6T bitcell called BCmin in the ViPro libary by placing PMOS_VTL devices and NMOS_VTL devices from the NCSU_Devices_FreePDK45 library. The bitcell will have minimum device sizing except for the pass gate width = 570n and pull up width = 220n..

darCreateSenseAmpSchematic()

```
darCreateSenseAmpSchematic(
        libName
        cellName
        FETlibName
        PMOScellName
        NMOScellName
        technologyCase
        cellName_nand
        @key (wen wnMin) (len lnMin) (weql wpMin) (leql wpMin) (wpsa wpMin) (lpsa lpMin)
        (wnsa wnMin) (lnsa lnMin) (wbl wnMin) (lbl lnMin) (wsapc wpMin) (lsapc lpMin)
        (wpNAND wpMin) (lpNAND lpMin) (wnNAND wnMin) (lnNAND lnMin)
        )
```

Description

Creates a sense amplifier with input pins enable and precharge (SAE and SAEPREC, respectively), input/output pins for the column-MUXed bit lines (RDWR and NRDWR), and output pin from the SR latch (SD). The procedure also requires the cell name of the 2-input NAND gate to be used in the SR latch. It is assumed the NAND gate is located in the same library as where the sense amp cell will be placed. The sense amp has the following parameters: wen, len, weql, leql, wpsa, lpsa, wnsa, lnsa, wbl, lbl, wsapc, lsapc, wpNAND, lpNAND, wnNAND, and lnNAND.

The abbreviations correspond to the following FETs in the schematic:
en - enable transistor (footer) (NMOS)
eql - equalize transistors for precharging the out/outb nodes of the sense amp (PMOS)
sa - transistors of the actual sense amp (PMOS and NMOS)
bl - transistors for BL/BLB inputs (NMOS)
sapc - precharge transistors for BL/BLB nodes (PMOS)
NAND - the NAND gate for the SR latch

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
FETlibName	String specifying the library where the FETs are located.
PMOScellName	String specifying the cell name of the PMOS FET to be placed.
NMOScellName	String specifying the cell name of the NMOS FET to be placed.
technologyCase	String specifying the letter required for the used technology.

cellName_nand	String specifying the cell name of the 2-input NAND gate to be used in the SR latch.
wen	String specifying the width of the enable (NMOS) FET (footer device).
len	String specifying the length of the enable (NMOS) FET (footer device).
weql	String specifying the width of the equalize (PMOS) FETs.
leql	String specifying the length of the equalize (PMOS) FETs.
wpsa	String specifying the width of the sense amp (PMOS) FETs.
lpsa	String specifying the length of the sense amp (PMOS) FETs.
wnsa	String specifying the width of the sense amp (NMOS) FETs.
lnsa	String specifying the length of the sense amp (NMOS) FETs.
wbl	String specifying the width of the input BL/BLB (NMOS) FETs.
lbl	String specifying the length of the input BL/BLB (NMOS) FETs.
wsapc	String specifying the width of the BL/BLB precharge (PMOS) FETs.
lsapc	String specifying the length of the BL/BLB precharge (PMOS) FETs.
wpNAND	String specifying the PMOS width of the SR NAND gate.
lpNAND	String specifying the PMOS length of the SR NAND gate.
wnNAND	String specifying the NMOS width of the SR NAND gate.
lnNAND	String specifying the NMOS length of the SR NAND gate.

Examples

darCreateSenseAmpSchematic(“ViPro” “SAmin” “NCSU_Devices_FreePDK45” “PMOS_VTL” “NMOS_VTL “upper”)
=> Creates a sense amp called SAmin in the ViPro libary by placing PMOS_VTL devices and NMOS_VTL devices from the NCSU_Devices_FreePDK45 library. The sense amp will have minimum device sizing because no parameters were specified.

darCreateSenseAmpSchematic(“ViPro” “SA” “NCSU_Devices_FreePDK45” “PMOS_VTL” “NMOS_VTL “upper” ?wen “375n ?wnNAND “450n”)
=> Creates a sense amp called SA in the ViPro libary by placing PMOS_VTL devices and NMOS_VTL devices from the NCSU_Devices_FreePDK45 library. The sense amp will have

minimum device sizing except for the enable FETs with width = 375n and the NMOS widths in the NAND-based SR latch = 450n.

darCreateSingleCDSchematic()

```
darCreateSingleCDSchematic(
        libName
        cellName
        FETlibName
        PMOScellName
        NMOScellName
        technologyCase
        @key (wpc wpMin) (lpc lpMin) (wpTX wpMin) (lpTX lpMin) (wnTX wnMin)
        (lnTX lnMin)
        )
```

Description
Creates a single column MUX, employing a transmission gate, with an input pin to select the column (CSEL and CSELB), an input pin to precharge the BL/BLB node (PCH), and input/output pins of bit lines from the SRAM (BL and BLB) and the MUXed bit lines (RDWR and NRDWR). There is 2 precharge transistors and one equalize transistor. The column MUX has the following parameters: wpc, lpc, wpTX, lpTX, wnTX, and lnTX.

The abbreviations correspond to the following FETs in the schematic:
pc - precharge/equalize transistors (PMOS)
TX - transmission gate transistors

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
FETlibName	String specifying the library where the FETs are located.
PMOScellName	String specifying the cell name of the PMOS FET to be placed.
NMOScellName	String specifying the cell name of the NMOS FET to be placed.
technologyCase	String specifying the letter required for the used technology.
wpc	String specifying the width of the precharge/equalize (PMOS) FETs.
lpc	String specifying the length of the precharge/equalize (PMOS) FETs.
wpTX	String specifying the width of the transmission gate PMOS FETs.
lpTX	String specifying the length of the transmission gate PMOS FETs.
wnTX	String specifying the width of the transmission gate NMOS FETs.

lnTX String specifying the length of the transmission gate NMOS FETs.

Examples

darCreateSingleCDSchematic(“ViPro” “SingleCDmin” “NCSU_Devices_FreePDK45” “PMOS_VTL” “NMOS_VTL “upper”)
=> Creates a single column MUX called SingleCDmin in the ViPro library by placing PMOS_VTL devices and NMOS_VTL devices from the NCSU_Devices_FreePDK45 library. The CD will have minimum device sizing because no parameter was specified

darCreateSingleCDSchematic(“ViPro” “SingleCD” “NCSU_Devices_FreePDK45” “PMOS_VTL” “NMOS_VTL “upper” ?wpTX “265n” ?wnTX “175n”)
=> Creates a single column MUX called SingleCD in the ViPro library by placing PMOS_VTL devices and NMOS_VTL devices from the NCSU_Devices_FreePDK45 library. The MUX has minimum device sizing except for the widths of the PMOS and NMOS of the transmission gate, with PMOS width = 265n and NMOS width = 175n.

darCreateMultiCDSchematic()

```
darCreateMultiCDSchematic(
        libName
        cellName
        FETlibName
        PMOScellName
        NMOScellName
        technologyCase
        MUX
        @key (wpc wpMin) (lpc lpMin) (wpTX wpMin) (lpTX lpMin) (wnTX wnMin)
        (lnTX lnMin)
        )
```

Description

Creates a multi column MUX, employing a transmission gate, with input pins to select the column (CSEL<0:MUX-1> and CSELB<0:MUX-1> - MUX is the amount of columns divided the word size), an input pin to precharge the BL/BLB node (PCH), and input/output pins of bit lines from the SRAM (BL<0:MUX-1> and BLB<0:MUX-1>) and the MUXed bit lines (RDWR and NRDWR). There is 2 precharge transistors and one equalize transistor. The column MUX has the following parameters: wpc, lpc, wpTX, lpTX, wnTX, and lnTX.

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
FETlibName	String specifying the library where the FETs are located.
PMOScellName	String specifying the cell name of the PMOS FET to be placed.
NMOScellName	String specifying the cell name of the NMOS FET to be placed.
technologyCase	String specifying the letter required for the used technology.
wpc	String specifying the width of the precharge/equalize (PMOS) FETs.
lpc	String specifying the length of the precharge/equalize (PMOS) FETs.
wpTX	String specifying the width of the transmission gate PMOS FETs.
lpTX	String specifying the length of the transmission gate PMOS FETs.
wnTX	String specifying the width of the transmission gate NMOS FETs.
lnTX	String specifying the length of the transmission gate NMOS FETs.

Examples

darCreateMultiCDSchematic(“ViPro” “MultiCDmin” “NCSU_Devices_FreePDK45” “PMOS_VTL” “NMOS_VTL “upper”)

=> Creates a multi column MUX called MultiCDmin in the ViPro library by placing PMOS_VTL devices and NMOS_VTL devices from the NCSU_Devices_FreePDK45 library. The circuit is similar to the single CD except that there are more CSEL/CSELB signals to choose which BL/BLB to pass through. The CD will have minimum device sizing because no parameter was specified.

darCreateMultiCDSchematic(“ViPro” “MultiCD” “NCSU_Devices_FreePDK45” “PMOS_VTL” “NMOS_VTL “upper” ?wpTX “265n” ?wnTX “175n”)

=> Creates a multi column MUX called MultiCD in the ViPro library by placing PMOS_VTL devices and NMOS_VTL devices from the NCSU_Devices_FreePDK45 library. The circuit is similar to the single CD except that there are more CSEL/CSELB signals to choose which BL/BLB to pass through. The MUX has minimum device sizing except for the widths of the PMOS and NMOS of the transmission gate, with PMOS width = 265n and NMOS width = 175n.

darChooseAndCreateCDSchematic()

```
darChooseAndCreateCDSchematic(
        libName
        cellName
        FETlibName
        PMOScellName
        NMOScellName
        technologyCase
        MUX
        @key (wpc wpMin) (lpc lpMin) (wpTX wpMin) (lpTX lpMin) (wnTX wnMin)
        (lnTX lnMin)
        )
```

Description

Invokes the correct procedure to create a column MUX depending on the number of columns (C) and the word size (DW), either a single column MUX or multi column MUX. This procedure passes the following parameters to the actual procedure that creates the column MUX: wpc, lpc, wpTX, lpTX, wnTX, and lnTX.

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
FETlibName	String specifying the library where the FETs are located.
PMOScellName	String specifying the cell name of the PMOS FET to be placed.
NMOScellName	String specifying the cell name of the NMOS FET to be placed.
technologyCase	String specifying the letter required for the used technology.
MUX	Integer specifying the number of MUXed lines required for the column MUX. This value is calculated by taking the number of columns divided by the length of a word in the SRAM.
wpc	String specifying the width of the precharge/equalize (PMOS) FETs.
lpc	String specifying the length of the precharge/equalize (PMOS) FETs.
wpTX	String specifying the width of the transmission gate PMOS FETs.
lpTX	String specifying the length of the transmission gate PMOS FETs.
wnTX	String specifying the width of the transmission gate NMOS FETs.

lnTX String specifying the length of the transmission gate NMOS FETs.

Examples

darChooseandCreateCDSchematic(“ViPro” “CD” “NCSU_Devices_FreePDK45” “PMOS_VTL” “NMOS_VTL “upper” 1)

=> Creates a single column MUX called CD in the ViPro library by placing PMOS_VTL devices and NMOS_VTL devices from the NCSU_Devices_FreePDK45 library. It is only a one column MUX because the word size is equal to the number of columns. The control signals for the columns are CSEL and CSELB. The circuit then follows the single column MUX procedure. See darCreateSingleCDSchematic() examples for further reference on device sizing.

darChooseandCreateCDSchematic(“ViPro” “CD” “NCSU_Devices_FreePDK45” “PMOS_VTL” “NMOS_VTL “upper” 4)

=> Creates a multi column MUX called CD in the ViPro library by placing PMOS_VTL devices and NMOS_VTL devices from the NCSU_Devices_FreePDK45 library. It is a multi column MUX because the number of columns is greater than word size. The control signals for the columns are CSEL<0:3> and CSELB<0:3>. The circuit then follows the multi column MUX procedure. See darCreateMultiCDSchematic() examples for further reference on device sizing.

darCreateDFFSchematic()

darCreateDFFSchematic(
libName
cellName
FETlibName
PMOScellName
NMOScellName
technologyCase
)

Description
Creates a D Flip Flop that has input pins for the clock and data input (CLK and D, respectively) and an output for the data that is stored (Q). The entire DFF has specific sizing. Therefore, there are no parameters that can be altered in this circuit.

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
FETlibName	String specifying the library where the FETs are located.
PMOScellName	String specifying the cell name of the PMOS FET to be placed.
NMOScellName	String specifying the cell name of the NMOS FET to be placed.
technologyCase	String specifying the letter required for the used technology.

Examples
darCreateDFFSchematic(“ViPro” “DFF” “NCSU_Devices_FreePDK45” “PMOS_VTL” “NMOS_VTL “upper”)
=> Creates a D Flip Flop called DFF in the ViPro library by placing PMOS_VTL devices and NMOS_VTL devices from the NCSU_Devices_FreePDK45 library.

darCreate2to1MUXSchematic()

```
darCreate2to1MUXSchematic(
        libName
        cellName
        cellName_inverter
        cellName_nand
        @key (wpNAND wpMin) (wnNAND wnMin) (lpNAND lpMin) (lnNAND lnMin)
        (wpINV wpMin) (wnINV wnMin) (lpINV lpMin) (lnINV lnMin)
        )
```

Description

Creates a 2:1 MUX with input pins for data (IN0 and IN1) and the select line (SELECT). The procedure requires the cell names of the inverter and NAND gates to be used. The 2:1 MUX has the following parameters: wpNAND, wnNAND, lpNAND, lnNAND, wpINV, wnINV, lpINV, and lnINV.

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
cellName_inverter	String specifying the cell name of the inverter to be used.
cellName_nand	String specifying the cell name of the 2-input NAND gate to be used.

Example

darCreate2to1MUXSchematic(“ViPro” “2to1” “INV” “NAND2”)
=> Creates a 2:1 MUX called 2to1 in the ViPro library. Both the inverter and NAND gates used are minimum sized since no parameters were specified.

darCreate2to1MUXSchematic(“ViPro” “2to1” “INV” “NAND2” ?wpINV “225n” ?wnNAND “300n”)
=> Creates a 2:1 MUX called 2to1 in the ViPro library. The inverter has PMOS width = 225n and the NAND gate has NMOS widths = 300n. All other devices are minimum sized because no other parameters were specified

darCreate4to1MUXSchematic()

darCreate4to1MUXSchematic(
libName
cellName
cellName_2to1MUX
)

Description
Creates a 4:1 MUX with input pins for data (IN0, IN1, IN2, and IN3) and the select lines (SELECT0 and SELECT1). The procedure requires the cell name of the 2:1 to be used. The 4:1 MUX has no parameters. The sizing must have been set when creating the 2:1 MUX.

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
cellName_2to1MUX	String specifying the cell name of the inverter to be used.

Example
darCreate4to1MUXSchematic(“ViPro” “4to1” “2to1”)
=> Creates a 4:1 MUX called 4to1 in the ViPro library. The sizing of the 2:1 MUXes used were sized when created; there are no sizing parameters to be altered.

darCreateTunableDelaySchematic()

darCreateTunableDelaySchematic(
libName
cellName
cellName_delay
cellName_4to1
)

Description
Creates a tunable delay element which is 4 identical delay elements in series. The output of each delay element is put into a 4:1 MUX which then chooses how much delay to use. There is an input pin for the signal to be delayed, input pins for the select lines of the MUX (SELECT0 and SELECT1), and an output pin of the MUX (OUT). The cell names of the delay element and 4:1 MUX to be used must be passed as arguments. To size the delay element, must first create a delay element (See darCreateDelayElementSchematic() for further reference). To size the 4:1 MUX, must call darCreate2to1MUXSchematic() and pass desired sizing, then pass the 2:1 MUX's cell name to darCreate4to1MUXSchematic()

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
cellName_delay	String specifying the cell name of the delay element to be used.
cellName_4to1	String specifying the cell name of the 4:1 MUX to be used.

Example
darCreateTunableDelaySchematic("ViPro" "TuneDelay" "DE" "4to1")
=> Creates a tunable delay element called TuneDelay in the ViPro library. The delay element DE is used for the four delays while the MUX 4to1 is used to MUX the four delayed signals. The sizing of the delay element and MUX was already set when those individual components were created.

darCreateIOSchematic()

darCreateIOSchematic(
libName
cellName
cellName_inverter
cellName_triState
cellName_DFF
@key (TSIwpINV wpMin) (TSIwnINV wnMin) (TSIwpEN wpMin) (TSIwnEN wnMin) (TSIlpINV lpMin) (TSIlnINV lnMin) (TSIlpEN lpMin) (TSIlnEN lnMin) (IOwpINV wpMin) (IOwnINV wnMin) (IOlpINV lpMin) (IOlnINV lnMin)
)

Description
Creates the Input/Output Block for the SRAM. This block contains two DFFs to store the input and output data value for a bit. There is also an inverter and two tri-state inverters to generate the signal to be placed on the bit lines during a write. In addition, there are two inverters to create the enable signals for the tri-state inverters. There are several input pins: data in signal (DIN<i>), two clock signals (ICLK and OCLK), the data bit to be stored (SD<i>) which comes from the output of the Sense Amp's SR latch, and a write enable signal (WEN). There are two input/ouput pins for the bit lines (RDWR<i> and NRWDR<i>). There is a single output pin for the data out signal (DOUT<i>). The procedure requires the cell names of the inverter, tri-state inverter, and DFF to be used within this block. The sizing of the inverter and tri-state inverters can be altered through the procedure parameters. The I/O Block has the following parameters: TSIwpINV, TSIwnINV, TSIwpEN, TSIwnEN, TSIlpINV, TSIlnINV, TSIlpEN, TSIlnEN, IOwpINV, IOwnINV, IOlpINV, and IOlnINV.

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
cellName_inverter	String specifying the cell name of the inverter to be used.
cellName_triState	String specifying the cell name of the tri-state inverter to be used.
cellName_DFF	String specifying the cell name of the DFF to be used.
TSIwpINV	String specifying the PMOS width of the inverter portion of the tri-state inverters.
TSIwnINV	String specifying the NMOS width of the inverter portion of the tri-state inverters.
TSIwpEN	String specifying the PMOS width of the enable portion of the tri-state inverters.

TSIwnEN	String specifying the NMOS width of the enable portion of the tri-state inverters.
TSIlpINV	String specifying the PMOS length of the inverter portion of the tri-state inverters.
TSIlnINV	String specifying the NMOS length of the inverter portion of the tri-state inverters.
TSIlpEN	String specifying the PMOS length of the enable portion of the tri-state inverters.
TSIlnEN	String specifying the NMOS length of the enable portion of the tri-state inverters.
IOwpINV	String specifying the PMOS width of the inverters.
IOwnINV	String specifying the NMOS width of the inverters.
IOlpINV	String specifying the PMOS length of the inverters.
IOlnINV	String specifying the NMOS length of the inverters.

Examples

darCreateIOSchematic(“ViPro” “IOBlock” “INV” “TSI” “DFF”)
=> Creates the input/output block called IOBlock in the ViPro library. It uses the inverter INV, the tri-state inverter TSI, and the D Flip Flop DFF within the block.

darCreateIOSchematic(“ViPro” “IOBlock” “INV” “TSI” “DFF” ?TSIwpINV 360n ?IOwnINV 95n)
=> Creates the input/output block called IOBlock in the ViPro library. It uses the inverter INV, the tri-state inverter TSI, and the D Flip Flop DFF within the block. The PMOS of the inverter portion of all tri-state inverters has width = 360 and the NMOS of all the inverters have width = 95n.

darCreateBufferChainSchematic()

```
darCreateBufferChainSchematic(
        libName
        cellName
        FETlibName
        PMOScellName
        NMOScellName
        technologyCase
        stages
        fanout
        @key (minWp wpMin) (minLp lpMin) (minWn wnMin) (minLn lnMin)
        )
```

Description

Creates a buffer chain by placing X inverters in series. X is a user specified input denoting the number of stages - how many inverters to place in series. The first inverter is minimum sized and all subsequent inverters are sized according to the fanout value specified by the user. The procedure only requires the minimum FET sizes for the given technology, whih are the following parameters: minWp, minLp, minWn, and minLn.

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
FETlibName	String specifying the library where the FETs are located.
PMOScellName	String specifying the cell name of the PMOS FET to be placed.
NMOScellName	String specifying the cell name of the NMOS FET to be placed.
technologyCase	String specifying the letter required for the used technology.
stages	String specifying the number of stages (inverters) in the buffer chain.
fanout	String specifying the fanout size for all subsequent stages afeter the first inverter
minWp	String specifying the minimum PMOS width for the given technology.
minLp	String specifying the minimum PMOS length for the given technology.
minWn	String specifying the minimum NMOS width for the given technology.
minLn	String specifying the minimum NMOS length for the given technology.

Examples
darCreateBufferChainSchematic(“ViPro” “BuffChain” “NCSU_Devices_FreePDK45” “PMOS_VTL” “NMOS_VTL “upper” “10” “4”)
=> Creates a buffer chain of 10 series inverters called BuffChain in the ViPro library by placing PMOS_VTL devices and NMOS_VTL devices from the NCSU_Devices_FreePDK45 library. The first inverter is minimum sized and all subsequent inverters are sized for a fan out of 4.

darCreateBufferChainSchematic(“ViPro” “BuffChain” “NCSU_Devices_FreePDK45” “PMOS_VTL” “NMOS_VTL “upper” “4” “1.5”)
=> Creates a buffer chain of 4 series inverters called BuffChain in the ViPro library by placing PMOS_VTL devices and NMOS_VTL devices from the NCSU_Devices_FreePDK45 library. The first inverter is minimum sized and all subsequent inverters are sized for a fan out of 1.5.

darCreateWLDriver8Schematic()

darCreateWLDriver8Schematic(
libName
cellName
cellName_inverter
cellName_nand
cellName_and
@key (wpInv wpMin) (wnInv wnMin) (lpInv lpMin) (lnInv lnMin) (wpNAND wpMin) (lpNAND lpMin) (wnNAND wnMin) (lnNAND lnMin) (wpANDnand wpMin) (lpANDnand lpMin) (wnANDnand wnMin) (lnANDnand lnMin) (wpANDinv wpMin) (lpANDinv lpMin) (wnANDinv wnMin) (lnANDinv lnMin)
)

Description
Creates the word line driver for 8 word lines by combining the outputs from the three predecoders.
The input pins are the 8 outputs from each of the 3 predecoders (PRE8_6<0:7>, PRE5_3<0:7>, WLclk<0:7>). There are also 2 input pins (A<0:7>, B<0:7>) which pluck off the correct output from the 2 most significant decoders to access the desired group of 8 word lines. There are 8 output pins for the word lines (WL<0:7>). The procedure requires the cell names of the NAND, AND, and inverter gates to be used. Sizing of each gate is accessible through the following parameters: wpInv, wnInv, lpInv, lnInv, wpNAND, lpNAND, wnNAND, lnNAND, wpANDnand, lpANDnand, wnANDnand, lnANDnand, wpANDinv, lpANDinv, wnANDinv, lnANDinv

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
cellName_inverter	String specifying the cell name of the inverter to be used.
cellName_nand	String specifying the cell name of the NAND gate to be used.
cellName_and	String specifying the cell name of the AND gate to be used.

Examples
darCreateWLDriver8Schematic(“ViPro” “WLDriver8” “INV” “NAND2” “AND2”)
=> Creates a word line driver for 8 word lines called WLDriver8 in the ViPro library. Since no parameters were specified, minimum sizing is used for the inverter, NAND, and AND gates. Th

darCreateWLDriver8Schematic(“ViPro” “WLD8” “INV” “NAND2” “AND2” ?wpInv “225n” ?wnANDnand “200n” ?wnANDinv “100n”)
=> Creates a word line driver for 8 word lines called WLD8 in the ViPro library. The inverter has PMOS width = 225n, the NMOSs in the NAND part of the AND gate has width = 200n, and

the NMOS in the inverter part of the AND gate has width = 100n. All other devices are minimum size since no other parameters were specified.

darCreateWLBufferSchematic()

darCreateWLBufferSchematic(
libName
cellName
R
cellName_WLBufferChain
)

Description
Creates the buffer chain ("k" chain) for all of the word lines, which allows for buffering up the signal coming from the word line driver. This buffered signal then goes into the bitcell array. The procedure requires the number of rows in the SRAM and the cell name of the already created "k" buffer chain. There are input pins for all the word lines (IN<0:R-1>) and output pins for all the word lines (OUT<0:R-1>).

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
R	Integer specifying the number of rows in the SRAM.
cellName_WLBufferChain	String specifying the cell name of the "k" buffer chain to be used.

Example
darCreateWLBufferSchematic("ViPro" "WLBuffer" 64 "KChain")
=> Creates a buffer chain called WLBuffer in the ViPro library for all 64 word lines. The buffer chain to be used (the "k" chain) should have been previously created and is called KChain. Thus, the sizing for the "k buffers" was set when the single "k chain" was created. This procedure only places the previously created "k" chain for the number of rows in the SRAM.

darCreatePREWLBufferSchematic()

darCreatePREWLBufferSchematic(
libName
cellName
cellName_PRE_WLBufferChain
)

Description
Creates the buffer chain ("n" chain) for all of the 8 outputs of the three 3:8 predecoders. These buffered signals then go into the word line driver. The procedure requires the cell name of the already created "n" buffer chain. There are input pins for all outputs of all the predecoders (PRE_PRE8_6<0:7>, PRE_PRE5_3<0:7>, and PRE_WLclk<0:7>) and output pins for buffered signals (PRE8_6<0:7>, PRE5_3<0:7>, and WLclk<0:7>).

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
cellName_PRE_WLBufferChain	String specifying the cell name of the "n" buffer chain to be used.

Example
darCreatePREWLBufferSchematic("ViPro" "PRE_WLBuffer" "Chain")
=> Creates a buffer chain called PRE_WLBuffer in the ViPro library for all 24 outputs from the three 3:8 predecoders. The buffer chain to be used (the "n" chain) should have been previously created and is called NChain. Thus, the sizing for the "n buffers" was set when the single "n chain" was created. This procedure only places the previously created "n chain" for the each output of all the predecoders.

darChoosePREWLDriverNode()

darChoosePREWLDriverNode(
n
)

Description
Returns an array with the names of the nodes after the three 3:8 decoders. The input is the "n" from the decoder output, the number of buffer stages after the three 3:8 decoders.

Arugments
n Integer specifying the number of "n buffers" after the predecoders.

Examples
darChoosePREWLDriverNode(0)
=> Returns the list ("PRE8_6<0:7>" "PRE5_3<0:7>" "WLclk<0:7>") to be used as the node names after the predecoders - no "n buffers" so it connects the predecoders' outputs straight into the word line driver.

darChoosePREWLDriverNode(3)
=> Returns the list ("PRE_PRE8_6<0:7>" "PRE_PRE5_3<0:7>" "PRE_WLclk<0:7>") to be used as the node names after the predecoders - there are three "n buffers" so it connects the predecoders' outputs into the buffers.

darChoosePREWLNode()

darChoosePREWLNode(
k
R
)

Description
Returns a string with the name of the nodes after the word line driver. The input is the number of rows in the SRAM and the "k" from the decoder output, the number of buffer stages after the word line driver.

Arguments

k	Integer specifying the number of "k buffers" after the word line driver.
R	Integer specifying the number of rows in the SRAM.

Examples
darChoosePREWLNode(2 128)
=> Returns the string "PRE_WL<0:127>" to be used as the node name after the word line driver - there are two "k buffers" so it connects the word line driver to the buffers.

darChoosePREWLNode(0 128)
=> Returns the string "WL<0:127>" to be used as the node name after the word line driver - no "k buffers" so it connects the word line driver straight into the bitcell Array.

darBasicSchematic.il

darUvaEceSchematicCreateInstParInverter()

```
darUvaEceSchematicCreateInstParInverter(
        cvid
        libName
        cellName
        Iname
        in
        out
        VDD
        VSS
        location
        @key (wp wpMin) (wn wnMin) (lp lpMin) (ln lnMin) (m 1)
        )
```

Description
Allows placement of a CDF parameterized inverter. This procedure allows the user to place multiple instances of the same inverter, but with different sizes, assuming it has been CDF parameterized. This prevents redundant creation of the same circuit. The procedure requires the cell name (of the schematic) that is being generated, the library where the invereter is located, the name to be given to this instance, the input pin, the output pin, the supply nets, and the location within the schematic. Optional parameters include the widths and lengths of the FETs of the inverter.

Arguments

cvid	String specifying the cell name of the schematic where the inverter will be placed.
libName	String specifying the library where the inverter to be placed is located.
cellName	String specifying the cell name of the inverter to be placed.
Iname	String specifying the name of the inverter, after it is placed in the schematic.
in	String specifying the input to the inverter.
out	String specifying the output of the inverter.
VDD	String specifying the power node for the inverter.
VSS	String specifying the ground node for the inverter.

location	Integers in coordinate form specifying the location to place the inverter within the schematic.
wp	String specifying the PMOS width of the inverter.
wn	String specifying the NMOS width of the inverter.
lp	String specifying the PMOS length of the inverter.
ln	String specifying the NMOS length of the inverter.
m	String specifying the multiplier of that instance within the schematic.

Examples
darUvaEceSchematicCreateInstParInverter(cvid “ViPro” “INV” “ITMNG_INV0” "WR" "NPRE_WEN" "VDD" "VSS" 7:-1.25)
=> Places a CDF parameterized inverter within the schematic defined by cvid (it is the schematic that is currently opened/being generated). The inverter is called INV taken from the ViPro library. Once placed within the schematic, it is given the name ITMNG_INV0. The input is WR and the output is NPRE_WEN. The power and ground nodes are connected to VDD and VSS, respectively. The inverter is placed at the x:y coordinate 7:-1.25 within the schematic. Since no parameters are specified, all FETs are set to minimum size.

darUvaEceSchematicCreateInstParInverter(cvid “ViPro” “INV” “ITMNG_INV2” "WEN" "WENB" "VDD" "VSS" 4:-2 ?wp “225n” ?wn “105n”)
=> Places a CDF parameterized inverter within the schematic defined by cvid (it is the schematic that is currently opened/being generated). The inverter is called INV taken from the ViPro library. Once placed within the schematic, it is given the name ITMNG_INV2. The input is WEN and the output is WENB. The power and ground nodes are connected to VDD and VSS, respectively. The inverter is placed at the x:y coordinate 4:-2 within the schematic. The PMOS has width = 225n and the NMOS has width = 105n.

darUvaEceSchematicCreateInstParNand2()

```
darUvaEceSchematicCreateInstParNand2(
        cvid
        libName
        cellName
        Iname
        inA
        inB
        out
        VDD
        VSS
        location
        @key (wp wpMin) (wn wnMin) (ln lnMin) (lp lpMin) (m 1)
        )
```

Description

Allows placement of a CDF parameterized 2-input NAND gate. This procedure allows the user to place multiple instances of the same NAND gate, but with different sizes, assuming it has been CDF parameterized. This prevents redundant creation of the same circuit. The procedure requires the cell name (of the schematic) that is being generated, the library where the NAND gate is located, the name to be given to this instance, the input pins, the output pin, the supply nets, and the location within the schematic. Optional parameters include the widths and lengths of the FETs of the NAND gate. The widths and lengths of the FETs are given to each FET; the procedure doesn't account for series transistors. For example, if wn is set to 125n, each NMOS within the NAND gate will have a width of 125n.

Arguments

cvid	String specifying the cell name of the schematic where the NAND gate will be placed.
libName	String specifying the library where the NAND gate to be placed is located.
cellName	String specifying the cell name of the NAND to be placed.
Iname	String specifying the name of the NAND, after it is placed in the schematic.
inA	String specifying one input to the NAND gate.
inB	String specifying the other input to the NAND gate.
out	String specifying the output of the NAND gate.
VDD	String specifying the power node for the NAND gate.

VSS	String specifying the ground node for the NAND gate.
location	Integers in coordinate form specifying the location to place the NAND gate within the schematic.
wp	String specifying the PMOS widths of the NAND gate.
wn	String specifying the NMOS widths of the NAND gate.
lp	String specifying the PMOS lengths of the NAND gate.
ln	String specifying the NMOS lengths of the NAND gate.
m	String specifying the multiplier of that instance within the schematic.

Examples

darUvaEceSchematicCreateInstParNand2(cvid “ViPro” “NAND2” “ITMNG_NAND0” “WR” “CLK” “PRE_WEN” “VDD” “VSS” 7:-2.5)

=> Places a CDF parameterized NAND gate within the schematic defined by cvid (it is the schematic that is currently opened/being generated). The NAND gate is called NAND2 taken from the ViPro library. Once placed within the schematic, it is given the name ITMNG_NAND0. The inputs are WR and CLK and the output is PRE_WEN. The power and ground nodes are connected to VDD and VSS, respectively. The NAND gate is placed at the x:y coordinate 7:-2.5 within the schematic. Since no parameters are specified, all FETs are set to minimum size.

darUvaEceSchematicCreateInstParNand2(cvid “ViPro” “NAND2” “ITMNG_NAND2” “A” “B” “OUT” “VDD” “VSS” 7:-2.5 ?wn “160n”)

=> Places a CDF parameterized NAND gate within the schematic defined by cvid (it is the schematic that is currently opened/being generated). The NAND gate is called NAND2 taken from the ViPro library. Once placed within the schematic, it is given the name ITMNG_NAND2. The inputs are A and B and the output is OUT. The power and ground nodes are connected to VDD and VSS, respectively. The NAND gate is placed at the x:y coordinate 7:-2.5 within the schematic. Both NMOS FETs have width = 160n. All other FETs are set to minimum size.

darUvaEceSchematicCreateInstParAnd2()

```
darUvaEceSchematicCreateInstParAnd2(
        cvid
        libName
        cellName
        Iname
        inA
        inB
        out
        VDD
        VSS
        location
        @key (wpNAND wpMin) (wnNAND wnMin) (wpINV wpMin) (wnINV wnMin)
        (lpNAND lpMin) (lnNAND lnMin) (lpINV lpMin) (lnINV lnMin) (m 1)
        )
```

Description

Allows placement of a CDF parameterized 2-input AND gate. This procedure allows the user to place multiple instances of the same AND gate, but with different sizes, assuming it has been CDF parameterized. This prevents redundant creation of the same circuit. The procedure requires the cell name (of the schematic) that is being generated, the library where the AND gate is located, the name to be given to this instance, the input pins, the output pin, the supply nets, and the location within the schematic. Optional parameters include the widths and lengths of the FETs of the AND gate. The widths and lengths of the FETs are given to each FET; the procedure doesn't account for series transistors. For example, if wn is set to 125n, each NMOS within the AND gate will have a width of 125n.

Arguments

cvid	String specifying the cell name of the schematic where the AND gate will be placed.
libName	String specifying the library where the AND gate to be placed is located.
cellName	String specifying the cell name of the AND to be placed.
Iname	String specifying the name of the AND, after it is placed in the schematic.
inA	String specifying one input to the AND gate.
inB	String specifying the other input to the AND gate.
out	String specifying the output of the AND gate.
VDD	String specifying the power node for the AND gate.

VSS	String specifying the ground node for the AND gate.
location	Integers in coordinate form specifying the location to place the AND gate within the schematic.
wpNAND	String specifying the PMOS widths of the NAND portion of the AND gate.
wnNAND	String specifying the NMOS widths of the NAND portion of the AND gate.
lpNAND	String specifying the PMOS lengths of the NAND portion of the AND gate.
lnNAND	String specifying the NMOS lengths of the NAND portion of the AND gate.
wpINV	String specifying the PMOS widths of the inverter portion of the AND gate.
wnINV	String specifying the NMOS widths of the inverter portion of the AND gate.
lpINV	String specifying the PMOS lengths of the inverter portion of the AND gate.
lnINV	String specifying the NMOS lengths of the inverter portion of the AND gate.
m	String specifying the multiplier of that instance within the schematic.

Examples

darUvaEceSchematicCreateInstParAnd2(cvid “ViPro” “AND2” “ITMNG_AND0” “WR” “DLY_WL” “PRE_WEN” “VDD” “VSS” 7:-2.5)

=> Places a CDF parameterized AND gate within the schematic defined by cvid (it is the schematic that is currently opened/being generated). The AND gate is called AND2 taken from the ViPro library. Once placed within the schematic, it is given the name ITMNG_AND0. The inputs are WR and DLY_WL and the output is PRE_WEN. The power and ground nodes are connected to VDD and VSS, respectively. The AND gate is placed at the x:y coordinate 7:-2.5 within the schematic. Since no parameters are specified, all FETs are set to minimum size.

darUvaEceSchematicCreateInstParAnd2(cvid “ViPro” “AND2” “ITMNG_NAND2” “A” “B” “OUT” “VDD” “VSS” 7:-2.5 ?wnNAND “160n” ?wpINV “195n”)

=> Places a CDF parameterized AND gate within the schematic defined by cvid (it is the schematic that is currently opened/being generated). The AND gate is called AND2 taken from the ViPro library. Once placed within the schematic, it is given the name ITMNG_AND2. The

inputs are A and B and the output is OUT. The power and ground nodes are connected to VDD and VSS, respectively. The NAND gate is placed at the x:y coordinate 7:-2.5 within the schematic. Both NMOS FETs of the NAND portion have width = 160n and the PMOS of the inverter portion has width = 195n. All other FETs are set to minimum size.

darUvaEceSchematicCreateInstParNor2()

```
darUvaEceSchematicCreateInstParNor2(
        cvid
        libName
        cellName
        Iname
        inA
        inB
        out
        VDD
        VSS
        location
        @key (wp wpMin) (wn wnMin) (ln lnMin) (lp lpMin) (m 1)
        )
```

Description

Allows placement of a CDF parameterized 2-input NOR gate. This procedure allows the user to place multiple instances of the same NOR gate, but with different sizes, assuming it has been CDF parameterized. This prevents redundant creation of the same circuit. The procedure requires the cell name (of the schematic) that is being generated, the library where the NOR gate is located, the name to be given to this instance, the input pins, the output pin, the supply nets, and the location within the schematic. Optional parameters include the widths and lengths of the FETs of the NOR gate. The widths and lengths of the FETs are given to each FET; the procedure doesn't account for series transistors. For example, if wp is set to 225n, each PMOS within the NOR gate will have a width of 125n.

Arguments

cvid	String specifying the cell name of the schematic where the NOR gate will be placed.
libName	String specifying the library where the NOR gate to be placed is located.
cellName	String specifying the cell name of the NOR to be placed.
Iname	String specifying the name of the NOR, after it is placed in the schematic.
inA	String specifying one input to the NOR gate.
inB	String specifying the other input to the NOR gate.
out	String specifying the output of the NOR gate.
VDD	String specifying the power node for the NOR gate.
VSS	String specifying the ground node for the NOR gate.

location	Integers in coordinate form specifying the location to place the NOR gate within the schematic.
wp	String specifying the PMOS widths of the NOR gate.
wn	String specifying the NMOS widths of the NOR gate.
lp	String specifying the PMOS lengths of the NOR gate.
ln	String specifying the NMOS lengths of the NOR gate.
m	String specifying the multiplier of that instance within the schematic.

Examples

darUvaEceSchematicCreateInstParNor2(cvid “ViPro” “NOR2” “ITMNG_NOR1” “CLK” “GEN_SAE” “X” “VDD” “VSS” 6:0)

=> Places a CDF parameterized NOR gate within the schematic defined by cvid (it is the schematic that is currently opened/being generated). The NOR gate is called NOR2 taken from the ViPro library. Once placed within the schematic, it is given the name ITMNG_NOR1. The inputs are CLK and GEN_SAE and the output is X. The power and ground nodes are connected to VDD and VSS, respectively. The NOR gate is placed at the x:y coordinate 6:0 within the schematic. Since no parameters are specified, all FETs are set to minimum size.

darUvaEceSchematicCreateInstParNor2(cvid “ViPro” “NOR2” “ITMNG_NOR3” “A” “B” “OUT” “VDD” “VSS” 7:-2.5 ?wp “375n”)

=> Places a CDF parameterized NOR gate within the schematic defined by cvid (it is the schematic that is currently opened/being generated). The NOR gate is called NOR2 taken from the ViPro library. Once placed within the schematic, it is given the name ITMNG_NOR3. The inputs are A and B and the output is OUT. The power and ground nodes are connected to VDD and VSS, respectively. The NOR gate is placed at the x:y coordinate 6:0 within the schematic. Since no parameters are specified, all FETs are set to minimum size. Both PMOS FETs have width = 375n. All other FETs are set to minimum size.

darUvaEceSchematicCreateInstParBuf()

```
darUvaEceSchematicCreateInstParBuf(
        cvid
        libName
        cellName
        Iname
        in
        out
        VDD
        VSS
        location
        @key (wp1 wpMin) (wn1 wnMin) (wp2 wpMin) (wn2 wnMin) (lp1 lpMin) (ln1 lnMin)
        (lp2 lpMin) (ln2 lnMin) (m 1)
        )
```

Description

Allows placement of a CDF parameterized buffer. This procedure allows the user to place multiple instances of the same buffer, but with different sizes, assuming it has been CDF parameterized. This prevents redundant creation of the same circuit. The procedure requires the cell name (of the schematic) that is being generated, the library where the buffer is located, the name to be given to this instance, the input pins, the output pin, the supply nets, and the location within the schematic. Optional parameters include the widths and lengths of the FETs of the buffer.

Arguments

cvid	String specifying the cell name of the schematic where the buffer will be placed.
libName	String specifying the library where the buffer to be placed is located.
cellName	String specifying the cell name of the buffer to be placed.
Iname	String specifying the name of the buffer, after it is placed in the schematic.
in	String specifying the input to the buffer.
out	String specifying the output of the buffer.
VDD	String specifying the power node for the buffer.
VSS	String specifying the ground node for the buffer.
location	Integers in coordinate form specifying the location to place the buffer within the schematic.

wp1	String specifying the PMOS width of the first inverter.
wn1	String specifying the NMOS width of the first inverter.
lp1	String specifying the PMOS length of the first inverter.
ln1	String specifying the NMOS length of the first inverter.
wp2	String specifying the PMOS width of the second inverter.
wn2	String specifying the NMOS width of the second inverter.
lp2	String specifying the PMOS length of the second inverter.
ln2	String specifying the NMOS length of the second inverter.
m	String specifying the multiplier of that instance within the schematic.

Examples

darUvaEceSchematicCreateInstParBuf(cvid “ViPro” “buffer” “IBUFCS<0>” “DEC_AC<0>” “DEC_ACB<0>” “VDD” “VSS” 3:2)

=> Places a CDF parameterized buffer within the schematic defined by cvid (it is the schematic that is currently opened/being generated). The buffer is called buffer taken from the ViPro library. Once placed within the schematic, it is given the name IBUFCS<0>. The input is DEC_AC<0> and the output is DEC_ACB<0>. The power and ground nodes are connected to VDD and VSS, respectively. The buffer is placed at the x:y coordinate 3:2 within the schematic. Since no parameters are specified, all FETs are set to minimum size.

darUvaEceSchematicCreateInstParBuf(cvid “ViPro” “buffer” “IBUFCS<2>” “DEC_AC<2>” “DEC_ACB<2>” “VDD” “VSS” 4:5 ?wp1 “185n” ?wn2 “95n”)

=> Places a CDF parameterized buffer within the schematic defined by cvid (it is the schematic that is currently opened/being generated). The buffer is called buffer taken from the ViPro library. Once placed within the schematic, it is given the name IBUFCS<2>. The input is DEC_AC<2> and the output is DEC_ACB<2>. The power and ground nodes are connected to VDD and VSS, respectively. The buffer is placed at the x:y coordinate 4:5 within the schematic. The PMOS of the first inverter has width = 185n and the NMOS of the second inverter has width = 95n. All other FETs are set to minimum size.

darUvaEceSchematicCreateInstParTriState()

darUvaEceSchematicCreateInstParTriState(
cvid
libName
cellName
Iname
in
triP
triN
out
VDD
VSS
location
@key (wpINV wpMin) (wnINV wnMin) (wpEN wpMin) (wnEN wnMin) (lpINV lpMin) (lnINV lnMin) (lpEN lpMin) (lnEN lnMin) (m 1)
)

Description
Allows placement of a CDF parameterized tri-state inverter. This procedure allows the user to place multiple instances of the same tri-state inverter, but with different sizes, assuming it has been CDF parameterized. This prevents redundant creation of the same circuit. The procedure requires the cell name (of the schematic) that is being generated, the library where the tri-state inverter is located, the name to be given to this instance, the input pins, the output pin, the supply nets, and the location within the schematic. Optional parameters include the widths and lengths of the FETs of the tri-state inverter, which have been grouped into enable FETs and the actual inverter FETs.

Arguments

cvid	String specifying the cell name of the schematic where the tri-state inverter will be placed.
libName	String specifying the library where the tri-state inverter to be placed is located.
cellName	String specifying the cell name of the tri-state inverter to be placed.
Iname	String specifying the name of the tri-state inverter, after it is placed in the schematic.
in	String specifying the input to the tri-state inverter.
triP	String specifying the enable signal to the PMOS enable FET of the tri-state inverter.

triN	String specifying the enable signal to the NMOS enable FET of the tri-state inverter.
out	String specifying the output of the tri-state inverter.
VDD	String specifying the power node for the tri-state inverter.
VSS	String specifying the ground node for the tri-state inverter.
location	Integers in coordinate form specifying the location to place the tri-state inverter within the schematic.
wpINV	String specifying the PMOS width of the actual inverter portion of the tri-state inverter.
wnINV	String specifying the NMOS width of the actual inverter portion of the tri-state inverter.
lpINV	String specifying the PMOS length of the actual inverter portion of the tri-state inverter.
lnINV	String specifying the NMOS length of the actual inverter portion of the tri-state inverter.
wpEN	String specifying the PMOS width of the enable portion of the tri-state inverter.
wnEN	String specifying the NMOS width of the enable portion of the tri-state inverter.
lpEN	String specifying the PMOS length of the enable portion of the tri-state inverter.
lnEN	String specifying the NMOS length of the enable portion of the tri-state inverter.
m	String specifying the multiplier of that instance within the schematic.

Examples

darUvaEceSchematicCreateInstParTriState(cvid “ViPro” “TSI” “TSI0” “dB” “NWEN” “WENB” “RDWR” “VDD” “VSS” 5.5:-1)

=> Places a CDF parameterized tri-state inverter within the schematic defined by cvid (it is the schematic that is currently opened/being generated). The tri-state inverter is called TSI taken from the ViPro library. Once placed within the schematic, it is given the name TSI0. The input is dB, the input to the PMOS enable FET is NWEN, the input to the NMOS enable FET is WENB, and the output is RDWR. The power and ground nodes are connected to VDD and VSS,

respectively. The buffer is placed at the x:y coordinate 5.5:-1 within the schematic. Since no parameters are specified, all FETs are set to minimum size.

darUvaEceSchematicCreateInstParTriState(cvid “ViPro” “TSI” “TSI2” “d” “NWEN” “WENB” “NRDWR” “VDD” “VSS” 5.5:-3 ?wpEN “325n” ?wnINV “170n”)
=> Places a CDF parameterized tri-state inverter within the schematic defined by cvid (it is the schematic that is currently opened/being generated). The tri-state inverter is called TSI taken from the ViPro library. Once placed within the schematic, it is given the name TSI2. The input is d, the input to the PMOS enable FET is NWEN, the input to the NMOS enable FET is WENB, and the output is NRDWR. The power and ground nodes are connected to VDD and VSS, respectively. The buffer is placed at the x:y coordinate 5.5:-3 within the schematic. The PMOS enable FET has width = 325n and the NMOS FET of the actual inverter portion has widht = 170n. All other FETs are set to minimum size.

darBitcellArray_Schem.il

darCreateArraySchematic()

darCreateArraySchematic(
    libName
    cellName
    rows
    columns
    cellName_bitcell
)

Description
Creates a bitcell array with the specified number of rows and columns. There are input pins for the word lines (WL<0:rows-1>) and innput/output pins for the bit lines (BL/BLB<0:columns-1>). The procedure requires the cell name of the bitcell to be used. To modify the sizing of the bitcell, must first create bitcell with desired sizing and pass its cell name to this procedure.

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
rows	Integer specifying the number of rows in the SRAM.
columns	Integer specifying the number of columns in the SRAM.
cellName_bitcell	String specifying the cell name of the bitcell to be used.

Example
darCreateArraySchematic(“ViPro” “BCArray_128x16” 128 16 “BC”)
=> Creates a 128 row by 16 column array of bitcells called BCArray_128x16 in the ViPro library. This array has the following pins: WL<0:127>, BL<0:15>, and BLB<0:15>.

darUvaEceWLDriver_Schem.il

darUvaEceCreateWLDriverSchematic()

darUvaEceCreateWLDriverSchematic(
libName
cellName
R
cellName_WLDriver8
)

Description
Creates the complete word line driver for as many rows there are in the bitcell array. There are input
pins for the outputs of the three predecoders, i.e. 24 inputs (PRE8_6<0:7>, PRE5_3<0:7>, WLclk<0:7>). There are output pins for every word line (WL<0:R-1>). The procedure requires the number of rows in the bitcell array and the cell name of the eight word line driver to be used. The procedure simply places as many eight word line drivers as needed to access every single row. Since this procedure is only placing instances of the eight word line driver, there are no parameters to be altered. The sizing must be altered when creating the eight word line driver. If there are N/K chains to buffer up the signal, the input and output node names are adjusted accordingly to account for the extra connections.

Arugments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
R	Integer that gives the number of rows in the bitcell array.
cellName_WLDriver8	String specifying the cell name of the eight word line driver to be used.

Examples
darUvaEceCreateWLDriverSchematic(“ViPro” “WLDriver_64” 64 “WLDriver8”)
=> Creates a word line driver for 64 rows.

darUvaEceCreateWLDriverSchematic(“ViPro” “WLDriver_128” 128 “WLDriver8”)
=> Creates a word line driver for 128 rows.

darUvaEceBitSlice_Schem.il

darUvaEceCreateBitSliceSchematic()

darUvaEceCreateBitSliceSchematic(
libName
cellName
cellName_SA
cellName_IO
cellName_CD
C
DW
)

Description
Creates the bitslice for the reading, writing, and storing circuit components. The bitslice contains three lower level components: the sense amplifier, the column MUX, and the Input/Output Block. This procedure connects all three componenets together. One column of components is a "slice"; there are as many slices as there are data bits for a word in the memory, i.e. for an 8 bit word, there will be 8 slices. The procedure requires the number of columns and the length of a word to determine how many control signals to use. Since this procedure only connects different circuit components, there are no parameters to be altered.

Arugments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
cellName_SA	String specifying the cell name of the sense amplifier to be used.
cellName_IO	String specifying the cell name of the input/output block to be used.
cellName_CD	String specifying the cell name of the cloumn MUX to be used.
C	Integer specifying the number of columns in the bitcell array.
DW	Integer specifying the length of a word in the memory.

Examples
darUvaEceCreateBitSliceSchematic("ViPro" "Bitslice_8" "SA" "IOBlock" "CD" 8 1)
=> Creates a bitslice called Bitslice_8 with 8 slices using the sense amp called SA, the input/output block called "IOBlock", and the column MUX called "CD". Because there are 8 columns and each word is only 1 bit long, all 8 select lines for the column MUX will be used.

darUvaEceCreateBitSliceSchematic(“ViPro” “Bitslice_8” “SA” “IOBlock” “CD” 16 4)
=> Creates a bitslice with 4 slices using the sense amp called SA, the input/output block called “IOBlock”, and the column MUX called “CD”. Because there are 16 columns and each word is 4 bits long, only 4 select lines are needed for the column MUX. Thus, CSEL/CSELB<0:3> will be used, and the other bits will be hanging (disconnected).

darUvaEceRowPredec_Schem.il

darUvaEceCreatePredecSchematic()

darUvaEceCreatePredecSchematic(
    libName
    cellName
    A
    cellName_inverter
    cellName_and
)

Description
Creates the three 3:8 predecoders for accessing the rows of the SRAM. The three predecoders are identical to each other. Each predecoder is made using a static AND implementation. All row address bits are inverted using a single inverter. The original address bits are not buffered and used "as is" when the pass through to the row predecoder circuit. Since there are three 3:8 predecoders, ViPro assumes a 9 bit address divided into three groups of 3 bits each. The AND implementation first combines the 2 LSB of any given group, then combines this result with the MSB of that group. Since there are three predecoders, for discussion purposes, the predecoder that decodes address bits <6:8> will be called the most significant predecoder, the predecoder that decodes address bits <3:5> will be called the middle predecoder, and the predecoder that decodes address bits <0:2> will be called the least significant predecoder. For the least signigicant predecoder, the outputs of are combined with a word line enable signal that allows the user to access a specific word line within a group of eight word lines. In the overall structure, the most significant predecoder provides access to a group of 64 word lines, the middle predecoder provides access to a group of 8 word lines within the 64 word lines, and the least significant predecoder provides access to a single word line within the group of 8 word lines. There are no parameters to be altered at this level in the hierarchy because ViPro assumes that all gates within the predecoder are minimum sized. However, provisions can easily be added to alter the gates within the predecoder by adding size arguments to the statements that instantiate the AND gates.

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
A	Integer specifying the number of address bits. For now, this argument should always be set to 9.
cellName_inverter	String specifying the cell name of the inverter to be used.
cellName_and	String specifying the cell name of the AND gate to be used.

Examples
darUvaEceCreatePredecSchematic(“ViPro” “preDecode” 9 “INV” “AND2”)
=> Creates the entire row predecoder called preDecode in the ViPro library using the inverter INV and AND gate AND2.

darUvaEceColDecoder_Schem.il

darUvaEceCreateColDecSchematic()

darUvaEceCreateColDecSchematic(
libName
cellName
cellName_inverter
cellName_and
)

Description
Creates a 3:8 column decoder for accessing the proper column in the SRAM. The decoder is made using a static AND implementation. All column address bits are inverted using a single inverter. The original address bits are not buffered and used "as is" when the pass through to the predecoder circuit, except for the MSB, which is ANDed with itself. The AND implementation first combines the 2 LSB then combines this result with the buffered MSB. There are no parameters to be altered at this level in the hierarchy because ViPro assumes that all gates within the column decoder are minimum sized. However, provisions can easily be added to alter the gates within the predecoder by adding size arguments to the statements that instantiate the AND gates.

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
cellName_inverter	String specifying the cell name of the inverter to be used.
cellName_and	String specifying the cell name of the AND gate to be used.

Examples
darUvaEceCreateColDecSchematic("ViPro" "colDecoder" "INV" "AND2")
=> Creates the column decoder called colDecoder in the ViPro libray using the inverter INV and the AND gate called AND2.

darTimingBlock_Schem.il

darCreateTimingBlockSchematic()

```
darCreateTimingBlockSchematic(
        libName
        cellName
        cellName_inverter
        cellName_buffer
        cellName_nand
        cellName_and
        cellName_nor
        cellName_dlyWLbuffChain
        cellName_ICLKbuffChain
        cellName_NSAPRECbuffChain
        cellName_WLENbuffChain
        cellName_SAEbuffChain
        cellName_WENbuffChain
        cellName_NPRECHbuffChain
        )
```

Description

Creates the entire timing block that ViPro requires/assumes. Since it is difficult to explain how everything is connected, the schematic of this circuit can be found in Appendix A. The timing block uses the typical inverter, NAND, AND, and NOR gates. There are 7 output pins for different control signals. Each of these signals are buffered within the timing block, which is why the procedure requires 7 different buffer chains. Each buffer chain could be of different length and different fanout. However, each buffer chain is generated the same way using darCreateBufferChainSchematic(). There are no parameters to be altered at this level in the hierarchy, but provisions can be added to to alter the basic gates within the timing block by adding size arguments to the statements that instantiate the inverter, buffer, AND, NAND, and NOR gates.

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
cellName_inverter	String specifying the cell name of the inverter to be used.
cellName_buffer	String specifying the cell name of the buffer to be used.
cellName_nand	String specifying the cell name of the NAND gate to be used.
cellName_and	String specifying the cell name of the AND gate to be used.

cellName_nor	String specifying the cell name of the NOR gate to be used.
cellName_dlyWLbuffChain	String specifying the cell name of the buffer chain to be used for the dlyWL signal.
cellName_ICLKbuffChain	String specifying the cell name of the buffer chain to be used for the ICLK signal.
cellName_NSAPRECbuffChain	String specifying the cell name of the buffer chain to be used for the NSAPREC signal.
cellName_WLENbuffChain	String specifying the cell name of the buffer chain to be used for the WLEN signal.
cellName_SAEbuffChain	String specifying the cell name of the buffer chain to be used for the SAE signal.
cellName_WENbuffChain	String specifying the cell name of the buffer chain to be used for the WEN signal.
cellName_NPRECHbuffChain	String specifying the cell name of the buffer chain to be used for the NPRECH signal.

Examples

darCreateTimingBlockSchematic(“ViPro” “timingBlock” “INV” “buffer” “NAND2” “AND2” “NOR2” “dlyWL_BChain” “ICLK_BChain” “NSAPREC_BChain” “WLEN_BChain” “SAE_BChain” “WEN_BChain” “NPRECH_BChain”)
=> Creates the timing block called timingBlock in the ViPro library using the inverter INV, the NAND gate called NAND2, the AND gate called AND2, the NOR gate called NOR2, and the following buffer chains for the control signals: “dlyWL_BChain”, “ICLK_BChain”, “NSAPREC_BChain”, “WLEN_BChain”, “SAE_BChain”, “WEN_BChain”, and “NPRECH_BChain”.

darUvaEcePredecTiming_Schem.il

darUvaEceCreateTimingSchematic()

darUvaEceCreateTimingSchematic(
libName
cellName
cellName_preDec
cellName_colDec
cellName_timing
cellName_buffer
cellName_nand
cellName_DFF
cellName_bufferChain
)

Description
Creates the timing and predecode block for the SRAM. It places the DFFs for the write signal, the row address bits, and the column address bits. It places the row predecoder, the column decoder, and the timing block. Finally, it combines the column decoder outputs with some control signals and buffers the resulting outputs. Since it is difficult to explain how everything is connected, the schematic of this circuit can be found in Appendix A. There are no parameters to be altered because this procedure merely connects lower level components together.

Arguments

libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
cellName_preDec	String specifying the cell name of the row predecoder to be used.
cellName_colDec	String specifying the cell name of the column decoder to be used.
cellName_timing	String specifying the cell name of the NAND gate to be used.
cellName_buffer	String specifying the cell name of the AND gate to be used.
cellName_nand	String specifying the cell name of the NOR gate to be used.
cellName_DFF	String specifying the cell name of the D Flip Flop to be used for the dlyWL signal.
cellName_bufferChain	String specifying the cell name of the buffer chain to be used for the ICLK signal.

Examples

darUvaEceCreateTimingSchematic(“ViPro” “TIMING_PREDEC” “preDecoder” “colDecoder” “timinBlock” “buffer” “NAND2” “DFF” “CSEL_BChain”)

=>Creates the predecode and timing block called TIMING_PREDEC in the ViPro library using the row predecoder preDecoder, the column decoder colDecoder, the timing block timingBlock, the buffer buffer, the DFF DFF, and the buffer chaing CSEL_BChain.

darUvaEceSchematic.il

```
darSchematicCreateFETInst(
        cvid
        libName
        cellName
        cellType
        instName
        leftPins
        rightPins
        topPins
        botPins
        origin
        rotation
        width
        length
        )
```

Description
Allows the user to place a transistor within the schematic. Similar to the procedure UvaEceSchematicCreateFETDefault except this procedure adds width and length properties to be read in as floats.

Agurments

cvid	String specifying cell view of the schematic.
libName	String specifying the library where the transistor is located.
cellName	String specifying the cell name of the transistor to be used.
cellType	String specifying the cell type to be used; this must be set to symbol.
instName	String specifying the name given to the transistor after it is placed in the schematic.
leftPins	List of strings specifying connections on the left side of the transistor.
rightPins	List of strings specifying connections on the right side of the transistor.
topPins	List of strings specifying connections on the top side of the transistor.
botPins	List of strings specifying connections on the bottom side of the transistor.
origin	String specifying the location to place the transistor in the schematic.
rotation	String specifying the rotation of the transistor within the schematic.

width Float specifying the width of the transistor.

length Float specifying the length of the transistor.

Examples

```
darSchematicCreateFETInst(cvid "NSCU_Devices_FreePDK45" "PMOS_VTL" "symbol" "P0"
        list(list(gate "IN"))
        list(list(bulk "VDD"))
        list(list(source "OUT"))
        list(list(drain "OUT"))
        2.5:-1 "R0" 180n 50n)
```

=> Places a transistor within the schematic defined by cvid. The transistor is the VT Low device called PMOS_VTL taken from the NSCU_Devices_FreePDK45 library. Once placed in the schematic, it is given the name P0. The transistor is placed at the x:y coordinate 2.5:-1 and is not rotated. The width is 180n and the length is 50n.

```
darSchematicCreateFETInst(cvid "NSCU_Devices_FreePDK45" "NMOS_VTH" "symbol"
        "N0"
        list(list(gate "IN"))
        list(list(bulk "VDD"))
        list(list(source "OUT"))
        list(list(drain "OUT"))
        0:2 "R0" 90n 50n)
```

=> Places a transistor within the schematic defined by cvid. The transistor is the VT High device called NMOS_VTH taken from the NSCU_Devices_FreePDK45 library. Once placed in the schematic, it is given the name N0. The transistor is placed at the x:y coordinate 0:2 and is not rotated. The width is 90n and the length is 50n.

darSchematicCreateFET()

darSchematicCreateFET(
cvid
libName
cellName
cellType
instName
leftPins
rightPins
topPins
botPins
origin
rotation
width
length
)

Similar to UvaEceSchematicCreateFETDefault except I added width and length properties to be read in as strings - This is used to instantiate a transistor that will be parameterized.

Description

Allows the user to place a transistor within the schematic. Similar to the procedure UvaEceSchematicCreateFETDefault except this procedure adds width and length properties to be read in as strings. This procedure is mainly used when placing a transistor within a schematic that will be CDF parameterized, i.e. inverter, buffer, NOR, NAND, etc.

Agurments

cvid	String specifying cell view of the schematic.
libName	String specifying the library where the transistor is located.
cellName	String specifying the cell name of the transistor to be used.
cellType	String specifying the cell type to be used; this must be set to symbol.
instName	String specifying the name given to the transistor after it is placed in the schematic.
leftPins	List of strings specifying connections on the left side of the transistor.
rightPins	List of strings specifying connections on the right side of the transistor.
topPins	List of strings specifying connections on the top side of the transistor.
botPins	List of strings specifying connections on the bottom side of the transistor.

origin	String specifying the location to place the transistor in the schematic.
rotation	String specifying the rotation of the transistor within the schematic.
width	String specifying the width of the transistor. This has been hard coded to be of the form wpx or wnx. The x may specify the type of gate it is, i.e. wpNAND represents the PMOS width of the NAND gate.
length	String specifying the length of the transistor. This has been hard coded to be of the form lpx or lnx. The x may specify the type of gate it is, i.e. lpNAND represents the PMOS length of the NAND gate.

Examples

```
darSchematicCreateFETInst(cvid "NSCU_Devices_FreePDK45" "PMOS_VTL" "symbol" "P0"
        list(list(gate "IN"))
        list(list(bulk "VDD"))
        list(list(source "OUT"))
        list(list(drain "OUT"))
        2.5:-1 "R0" "wpNAND" "lpNAND")
```

=> Places a transistor within the schematic defined by cvid. The transistor is the VT Low device called PMOS_VTL taken from the NSCU_Devices_FreePDK45 library. Once placed in the schematic, it is given the name P0. The transistor is placed at the x:y coordinate 2.5:-1 and is not rotated. The width is set to the variable wpNAND and the length is set to the variable lpNAND. This allows for the eventual NAND gate to be CDF parameterized.

```
darSchematicCreateFETInst(cvid "NSCU_Devices_FreePDK45" "NMOS_VTH" "symbol"
        "N0"
        list(list(gate "IN"))
        list(list(bulk "VDD"))
        list(list(source "OUT"))
        list(list(drain "OUT"))
        0:2 "R0" "wnEN" "lnEN")
```

=> Places a transistor within the schematic defined by cvid. The transistor is the VT High device called NMOS_VTH taken from the NSCU_Devices_FreePDK45 library. Once placed in the schematic, it is given the name N0. The transistor is placed at the x:y coordinate 0:2 and is not rotated. The width is set to the variable wnEN and the length is lnEN. This allows for the eventual tri-state inverter to be CDF parameterized.

darChooseTerminalCase()

```
darChooseTerminalCase(
        technologyCase
        )
```

Description
Returns an array with the letters for the FET terminals (drain, gate, source, and bulk) in either upper or lower case, depending on the requirement of the technology.

Arguments

technologyCase	String containing the letter case required for a given technology, only “upper” or “lower”.

Example
darChooseTerminalCase(“upper”)
=> Returns the following array for FET terminals: (“D”, “G”, “S”, “B”).

darChooseTerminalCase(“lower”)
=> Returns the following array for FET terminals: (“d”, “g”, “s”, “b”).

storeData()

storeData(
fileName
)

Description
Returns a hash map that stores the data from a text file. This procedure is normally used for reading in parameter values. The text file must be in the specific format that each line contains two sets of data, a parameter name and the parameter value. In addition, there must be one blank line at the end of the text file for the procedure to store the data properly. This procedure stores the data in hash map, called an association table in the Cadence environment. The first data in each line (the parameter name) is the key that will point to the second data (the parameter value) which is the value.

Arguments

fileName	String specifying the full path to the file to be read. The specified file path can also be relative to the folder where Cadence is run from, but using the full file path will ensure no errors in reading the file.

Examples
minSizeTable = storeData("./skill/minSizes.txt")
=> Stores the data found in the text file called "minSizes.txt" located in a folder called SKILL which is located in a sub-folder from where Cadence is run from. The text file contains 4 lines of data of the minimum FET sizes for the given technology node. The text file looks something like the following:

wpMin 180n
wnMin 90n
lpMin 50n
lnMin 50n
(blank line with no text)

The resulting hash map is called minSizeTable and has 4 keys: "wpMin", "wnMin", "lpMin", and "lnMin". These four keys point to the respective numerical value as depicted in the sample text file

sizes = storeData("/var/home/user/sizes.txt")
=> Stores the data found in the text file called "sizes.txt" located in the top most directory of a user. The text file contains 4 lines of data of the minimum FET sizes for the given technology node. The text file looks something like the following:

ICLKstages 6
ICLKFanOut 1.5
wpSA 180n
wpu 275n

wpg 125n
(blank line with no text)

The resulting hash map is called sizes and has 5 keys: “ILCKstages”, “ICLKFanOut”, “wpSA”, “wpu”, and “wpg”. These five keys point to the respective numerical value as depicted in the sample text file

darUvaEceSRAM_Schem.il

UvaEceCreateSRAMSchematic()

UvaEceCreateSRAMSchematic(
libName
cellName
R
C
DW
n
k
)

Description
Creates the entire SRAM schematic. Connects the leaf nodes together: Bitcell Array, WLDriver, Predecode & Timing, Bitslice. Also connects the "n" and "k" buffer chains, if necessary. The procedure properly accounts for node names depending if there are n/k buffer chains. There are input pins for the row address bits, the column address bits, the data input bits, the clock signal, and the write signal. There are input/output pins for the the data output bits. There are no parameters to alter at this level in the hierarchy as this procedure merely connects all the leaf nodes together.

Arguments

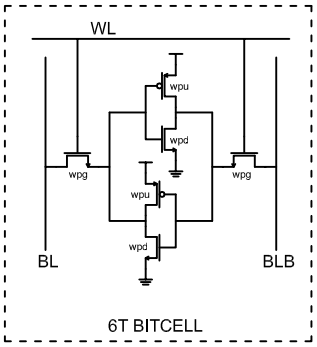
libName	String specifying the library to place the cell.
cellName	String specifying the name of the cell.
R	Integer specifying the number of rows in the SRAM.
C	Integer specifying the number of columns in the SRAM.
DW	Integer specifying the length of a word.
n	String specifying the number of "n buffers" after the predecoders.
k	String specifying the number of "k buffers" after the word line driver.

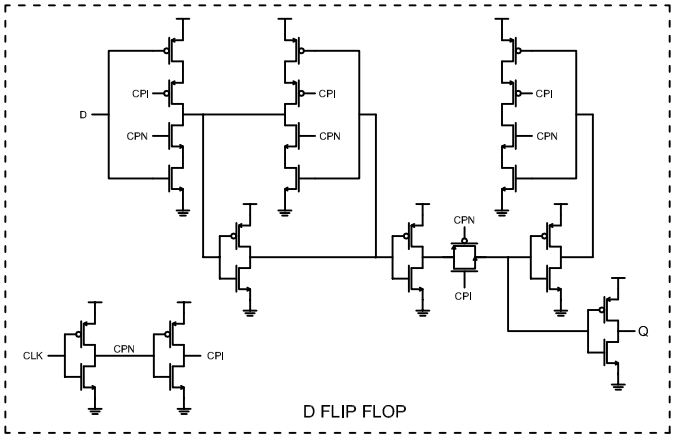
Examples
UvaEceCreateSRAMSchematic("ViPro" "SRAM_64x16" 64 16 4 "0" "0")
=> Creates the complete SRAM schematic called SRAM_64x16 in the ViPro library. There are 64 rows and 16 columns in the bitcell array. A word is 4 bits long. There are no "n" nor "k" buffer chains in this SRAM.

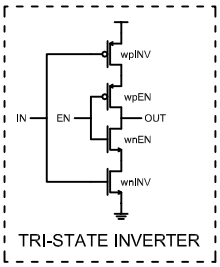
UvaEceCreateSRAMSchematic(“ViPro” “SRAM_64x16” 64 16 4 “2” “0”)
=> Creates the complete SRAM schematic called SRAM_64x16 in the ViPro library. There are 64 rows and 16 columns in the bitcell array. A word is 4 bits long. There is an “n” buffer chain, but no “k” chain.

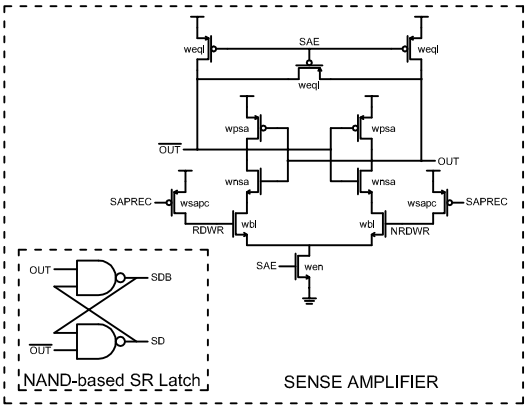
UvaEceCreateSRAMSchematic(“ViPro” “SRAM_64x16” 64 16 4 “4” “6”)
=> Creates the complete SRAM schematic called SRAM_64x16 in the ViPro library. There are 64 rows and 16 columns in the bitcell array. A word is 4 bits long. There is both “n” and “k” buffer chains in this SRAM.

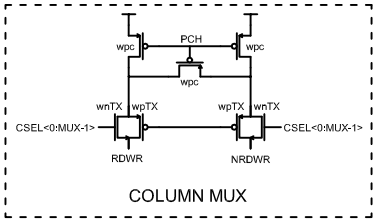
Appendix A: Schematics

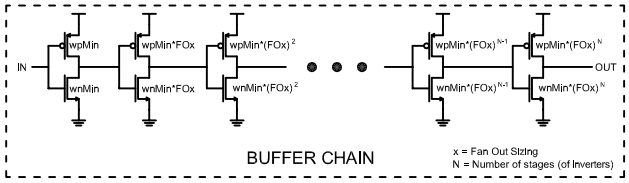




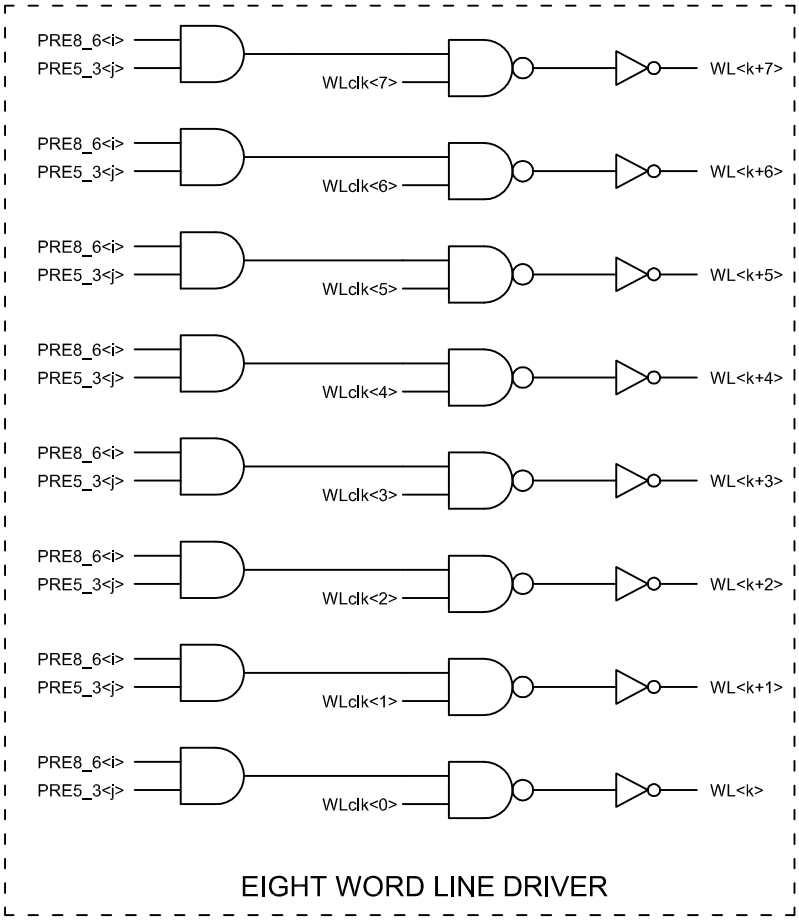


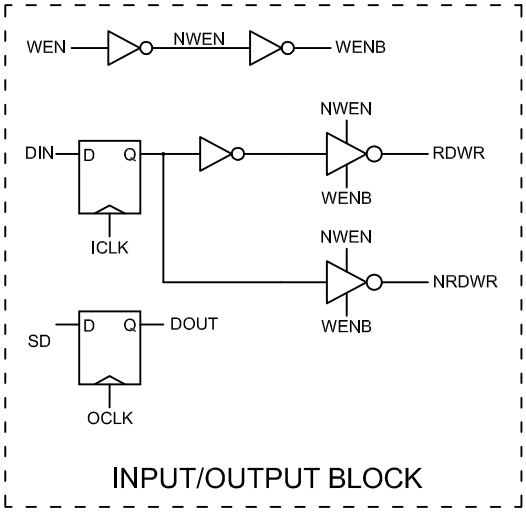




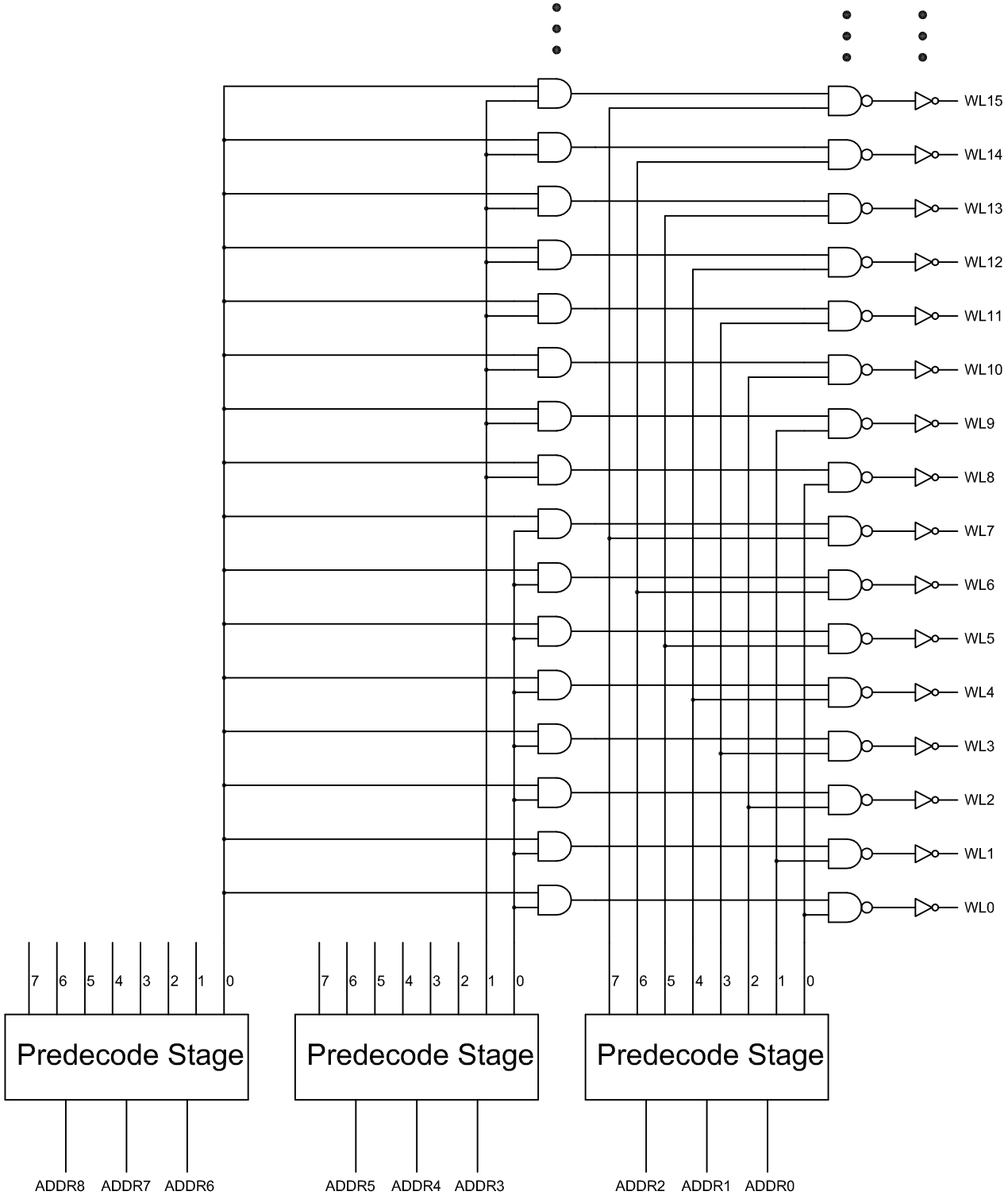


Appendix A: Schematics



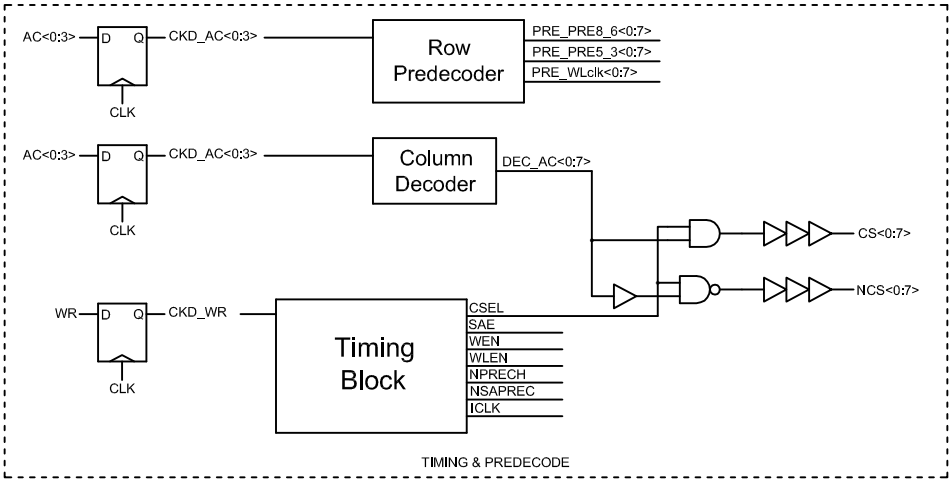


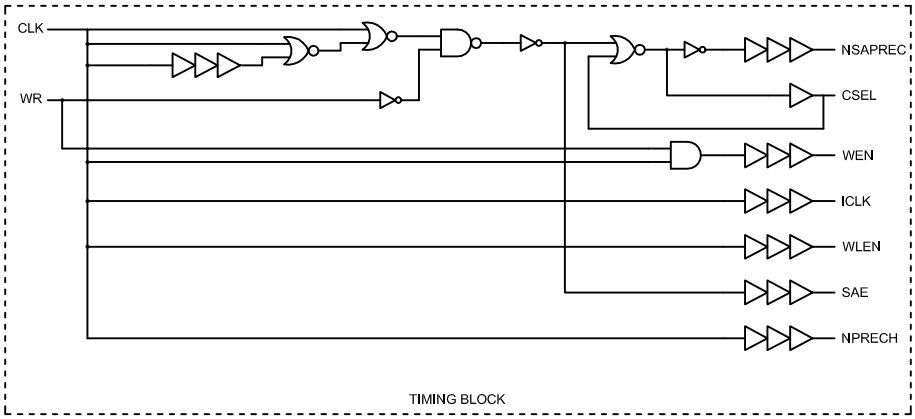
Appendix A: Schematics

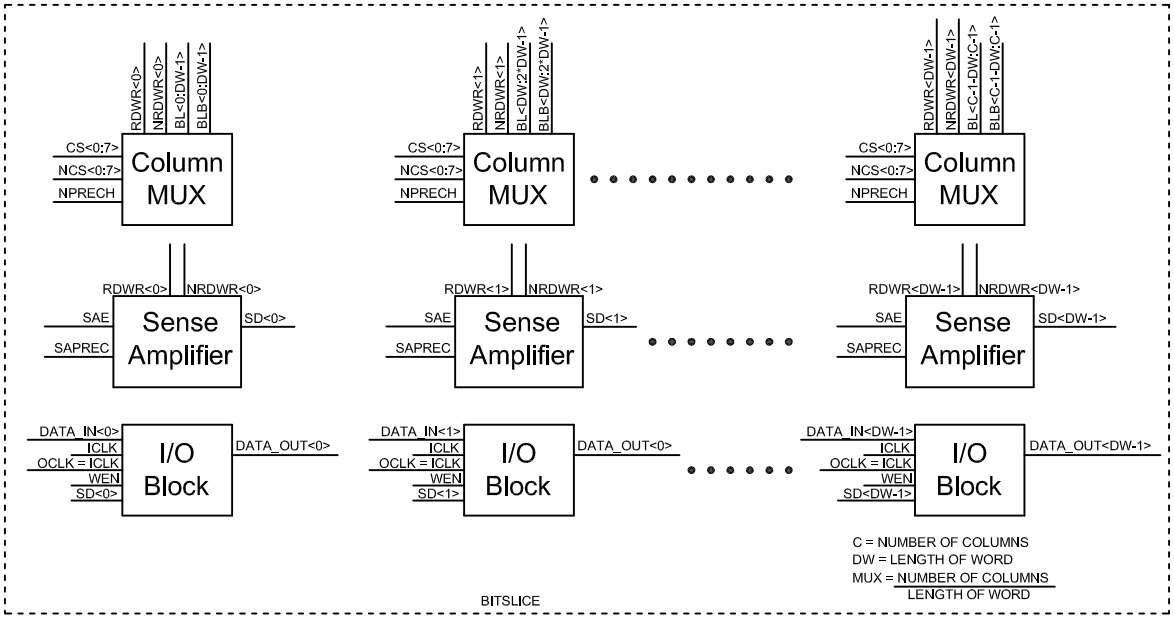


EIGHT WORD LINE DRIVER SHOWING COMBINATION OF PREDECODER OUTPUTS

Appendix A: Schematics







Appendix A: Schematics